Stealth Databases: Ensuring User-Controlled Queries in Untrusted Cloud Environments

Josef Spillner*‡, Martin Beck†, Alexander Schill*, Thomas Michael Bohnert‡
Technische Universität Dresden, Faculty of Computer Science
*Cloud Storage Lab (lab.nubisave.org), †Chair of Privacy and Data Security & BIOTEC
01062 Dresden, Germany
Email: {martin.beck1,alexander.schill}@tu-dresden.de
‡Zurich University of Applied Sciences, School of Engineering
Service Prototyping Lab (blog.zhaw.ch/icclab/)
8401 Winterthur, Switzerland
Email: {josef.spillner,thomas.bohnert}@zhaw.ch

***Abstract*—Sensitive data is increasingly being hosted online in ubiquitous cloud storage services. Recent advances in multi-cloud service integration through provider multiplexing and data dispersion have alleviated most of the associated risks for hosting files which are retrieved by users for further processing. However, for structured data managed in databases, many issues remain, including the need to perform operations directly on the remote data to avoid costly transfers. In this paper, we motivate the need for distributed *stealth databases* which combine properties from structure-preserving dispersed file storage for capacity-saving increased availability with emerging work on structure-preserving encryption for on-demand increased confidentiality with controllable performance degradation. We contribute an analysis of operators executing in map-reduce or map-carry-reduce phases and derive performance statistics. Our prototype, StealthDB, demonstrates that for typical amounts of personal structured data, stealth databases are a convincing concept for taming untrusted and unsafe cloud environments.**

I. Introduction and Definitions

Companies and individuals are for various reasons entrusting sensitive information to non-trustworthy services. Medical records, tax declarations, bills of material and planning documents are among the diverse kinds of data. This is a trend with both benefits and risks. In general, the risks include a lack of trust (what happens with the data, who else can access it) and robustness (will the user be able to access the data even in case of failures). Research on cloud storage integration systems advocates for a division of data across multiple independent storage services with appropriate coding and redundancy (*dispersed storage*) so that the loss of individual services can be tolerated and all providers would need to cooperate to access the entire data without user consent [1]. In practice, such information dispersal and secret sharing schemes offer sufficient safeguarding against eavesdropping and service unavailability, especially when being combined with encryption schemes.

However, storing data remotely and still processing it locally does generally not constitute a satisfactory design due to a high volume of data transmission and the concentration of load on a single client-side system. Typical data processing and analytics scenarios include ‘needle in a haystack’ string searches which yield a few records out of gigabytes of data, and statistical calculations over streams of sensor data. Ultimately, the user of the application needs to take a decision on the trade-off between confidentiality, availability, capacity and performance resulting from the coding parameterisation.

In order to overcome the limitations of untrusted and low-quality environments, *dispersed computing* concepts have been designed to distribute data storage, transmission and processing across any number of infrastructure services with a selected degree of redundancy for higher availability [2]. Assuming the non-cooperation of service providers, these concepts also increase the privacy of users.

Stealth computing extends dispersed computing by a stronger notion of confidentiality and privacy preservation. This concept is supposed to protect against the knowledge of what is being processed where, how and by whom. Historically, stealth software has been associated with a negative connotation mostly due to stealth computer viruses (malware) which are hard to detect [3]. Given recent major concerns about user privacy in cloud environments [4], there is a legitimate need for users to apply the stealth principles to protect their own activities in the cloud. The stealthiness is achieved by careful coding, in particular, encrypting the dispersed data fragments in a way that their structure is preserved to the extent that operations can still be performed on them. Fig. 1 visualises the general principle of stealth coding.

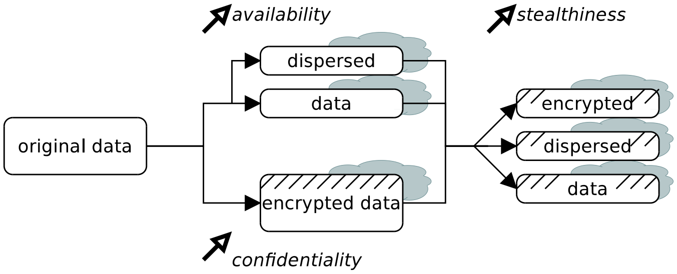

Fig. 1. Stealth coding principle: Combined (chained) coding for combined protection goals

However, while proper coding is essential, it is not a sufficient protection in dynamic environments such as public cloud services. Instead, dynamic reconfigurability of the data fragment location over multiple resource services and adaptive processing depending on the service capabilities need to be considered as well. Fig. 2 conveys the main idea of stealth computing in the context of service evolution over time. Instead of letting applications handle typical security and other evolution risks with individual error-prone methods, a lightweight stealth layer is made available to the application as a library or proxy service. The interfaces of the layer correspond to typical platform-level middleware services, such as file and object storage, databases, message queues and execution runtimes.

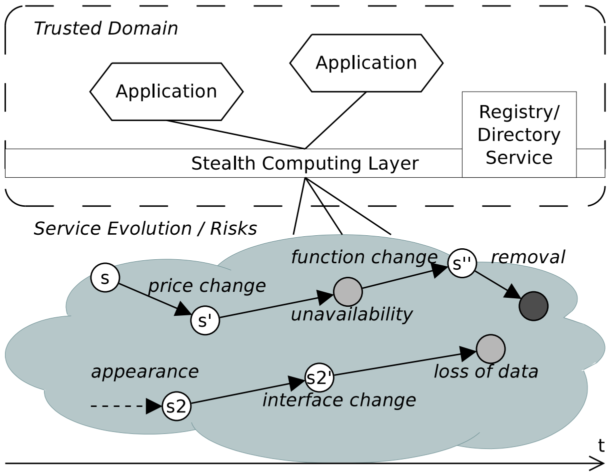

Fig. 2. Stealth layer to protect against the risks of data-centric service evolution over time

We could not find prior stealth computing approaches and hence have explored the design and behaviour of a stealth database system. The associated research challenges are (1) to design appropriate operators which can process stealth-coded data and (2) to find sweet spots in the trade-off between the expected slow-down and the transmission savings. In the following, we report on using stealth computing concepts within relational and non-relational column-store database architectures. First, we introduce concepts for a stealth database system which works in untrusted and unstable cloud environments. Then, we describe an implementation of such a system and explain how it performs and how it solves the research questions. Finally, we conclude with a critical review of our work and a comparison against related systems.

II. Stealth Database Concepts

A *stealth database* is a database management system which distributes its data and processing across several local resources as well as cloud storage and compute services with minimum redundancy in a way that neither the data nor intermediate processing results can be meaningfully interpreted by an adversary who gains read-only access to any subset of the services or full control over the redundant subset. Furthermore, despite service changes over time and unavailable services, stealth databases guarantee access to the data and responsiveness to queries with graceful degradation of accuracy, precision or completeness in the presence of service problems.

This *stealthiness* property is achieved by combining data coding and distribution methods to increase the confidentiality, integrity and availability of the data in unsafe and dynamically evolving cloud computing environments. In particular, information dispersal techniques with selective redundancy are combined with query-aware and privacy-preserving encryption techniques. For the information dispersal, bitsplitting is used due to its structure-preserving nature, which makes the resulting data fragments suitable for many arithmetic and analytical algorithms [5]. Each data value is split into k significant and m redundant fragments, for a total of $n = k + m (k \geq 1, m \geq 0)$. For the encryption, both homomorphic encryption (HE) [6] and order-preserving encryption (OPE) [7], [8] are employed on numeric data types to ensure that arithmetics and order-dependent queries can be performed on the encrypted data. For example, a range-aggregate query `SELECT SUM(x) FROM table WHERE x < 5` requires the column `x` to be encrypted in both an HE and an OPE cryptosystem because HE is not order-preserving by design and OPE does not support arithmetics. Furthermore, searchable encryption (SE) is employed on text data types. The order of coding mandates dispersion before encryption in order to maintain the value structures.

To support the distribution of individual value fragments, stealth databases take advantage of a relational *sub-column store* design where each column is dispersed across several resource services. For improved flexibility, we propose a resource/service multiplexing so that each column can be wholly or partially located in memory, on disk or in the cloud. Furthermore, it is assumed that a trustworty client application is available to interact with the untrusted cloud environment through sessions and to keep (minimal) table and keys state, and that cloud providers obey to the honest-but-curious threat model [9] although provisions are made to protect against byzantine failures. Trust may differ in each cloud provider; we assume that if it matters, it can be quantified as a non-functional property and used in the fragment distribution target selection and scheduling [10].

The following two subsections describe the storage layer of stealth database, i.e. the coding and distribution of data to various storage targets, and the processing layer, i.e. placement of database operators as close as possible to the data. Afterwards, type-dependent coding and the optimisation of insertions and queries regarding user-defined non-functional properties are elaborated on.

A. Data Coding, Distribution and Storage Combinations

A stealth database needs to function in the presence and absence of appropriate storage services. Therefore, its storage and access model forms a hybrid combination of an in-memory database which stores data in RAM, an embedded database which stores in files, and a distributed system which stores in appropriate services with and without compute capabilities.

1) Memory: Data is stored inside data structures in RAM and either discarded or persisted when a session ends.
2) File: Data is stored in one file per column or more efficient binary file structures.
3) Cloud: Data is sent to and retrieved from services with at least a CRUD (create, read, update, delete) storage interface.

Apart from the storage, the design also mandates where calculations take place. Typical database calculations are aggregations, filters, sorts and user-defined functions as well as relations across columns. Fig. 3 expresses the combination of memory, file and cloud databases. While calculations in the embedded database configurations (memory and file) as well as pure cloud storage configurations happens centrally, for full cloud compute services there is a choice to perform remote calculations depending on both the type of data and any prior coding and modification of its representation. The proposed programming model is map-reduce. When the mapping leads to ambiguous results due to missing entropy in sub-column value fragments, an iterative map-carry-reduce method with a handover protocol between the services needs to be used which effectively limits the parallel operator execution [5].

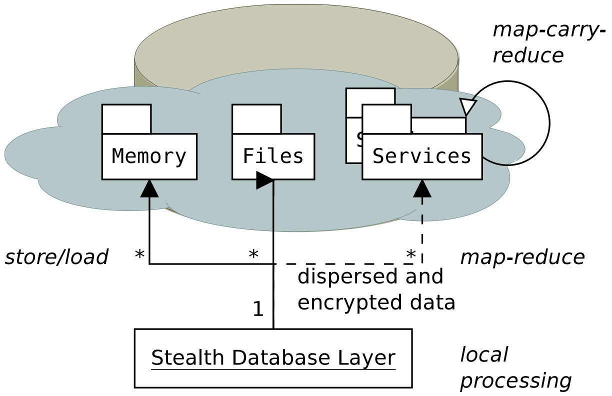

Fig. 3. Stealth database design with multiple storage locations

The data is distributed among the storage locations in a specified manner. The amount of redundancy influences the availability of data and the reliability of calculation results.

1) Random, round-robin or selective single placement: Data is stored at a single location, potentially depending on availability or capacity constraints, i.e. $k = 1, m = 0$.
2) Selective or weighted replication: Data is replicated a number of times and all replicas are placed according to algorithms such as balanced hash rings with $k < n, m = 0$ [11].
3) Full replication: Data is stored in all locations in parallel, leading to, for instance, triple replication with $k > 1$ and $m = 0$.
4) Dispersion or partial replication: Data is split into multiple fragments with a degree of redundancy less than a full replica, i.e. $m > 0$.

While replication offers a good trade-off between a low number of nodes and a high availability, the strength of dispersion is in the trade-off between low capacity and a higher number of tolerated node failures. Multiple dispersion schemes exist, out of which only a few lead to fragments that can be interpreted individually.

1) Erasure coding: Data is erasure-coded with a certain amount of redundancy and then split and stored. Fragments from the significant number of nodes are needed to interpret the data.
2) Secret sharing: Leads to fragments of the size of the original data. All fragments are needed to interpret the data.
3) Bitsplit coding: Similar to erasure coding, but using bitsplitting to allow for subsequent calculation on the data [5].
4) Content-specific fragments: For image files, the images are split visually and missing fragments can be interpolated.

Fig. 4 shows the difference between first three of the distribution modes for both a string value with five character bytes and an integer value which is represented as two bytes.

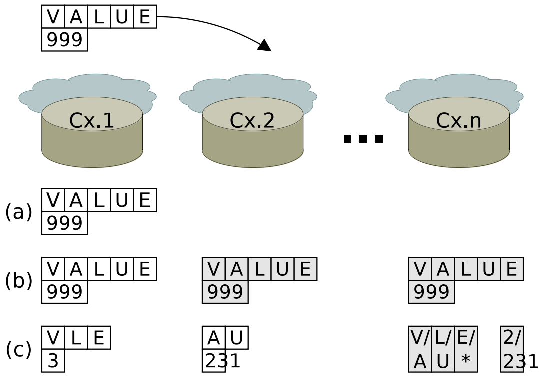

Fig. 4. Distribution modes: (a) random/round-robin/selective replication, (b) full replication, (c) bitsplit-coded with redundancy, fragment size is 8 bit

Generally, the following data types can be stealth-coded in a meaningful way:

1) Integer numbers.
2) Floating point numbers, by conversion to fixed-point numbers.
3) Strings.
4) Complex types, by decomposition into a combination of simple types.

In order to protect the data against additional security threats, in particular confidentiality violations, the data needs to be encrypted. Again, various options need to be considered.

1) Plain: Data is stored without encryption.
2) Conventional encryption: Symmetric encryption, requiring to store the key on the client side and fetching all data for processing.
3) Convergent encryption: Symmetric, using the hash value of the data as key. Needed for de-duplication of encrypted values.

4) Homomorphic encryption: Asymmetric encryption which allows for secure remote calculation. Keys need to be stored by the client with public keys propagated to each location.
5) Order-preserving encryption: Symmetric. Needed to support remote order-dependent calculations such as sorting.
6) Searchable encryption: Symmetric stream encryption with a private key and an initialisation vector. Needed to perform pattern and regular expression search over encrypted data.

The combination of operation-, distribution- and type-aware data coding shall be called *stealth coding* for it is a core enabler of stealth computing in addition to runtime facilities.

Additional meta-data about the data needs to be kept on demand to extend the support to a broader set of operations, and additional modifications may be applied to the data to favour non-functional properties other than security, such as performance or capacity. Similar to different encryption schemes, multiple modifications can be combined.

1) Simple: Data is stored without additional meta-data or modifications.
2) Ordering: Meta-data about the absolute position of a value in a column. Used for combination with OPE and for retrieval from selectively or randomly distributed data when guarantees about the order are essential.
3) Compression: Reduction of the capacity requirements, but also restriction of the operations which can be performed.
4) Redundancy: Configuration of a minimum level of redundancy.

The proposed configurability encompasses 4 base codings, $2^3 + 1 = 9$ coding modification combinations, $2^3 + 2 = 10$ encryption combinations and 4 base distributions. The combination of all coding, distribution and storage location choices leads to already 4320 configurations for only two nodes. Only a part of the configuration space will be suitable for remote processing over the resulting data. Therefore, the database system needs to be adaptive in choosing the right processing model for each operation in any given distributed data context.

B. Data Processing, Mapping and Reduction Combinations

Depending on the combination of location, distribution, coding and modification, the data processing workflow is transparently adapted to achieve a high data management quality in unsafe cloud environments. The quality is determined by the application preferences so that users have the possibility to exploit the data distribution. Whenever data fragments are stored in a compute cloud, either unencrypted or with an encryption scheme interpretable for an incoming query, then the associated query part is performed through the compute service as much as possible.

The query may have to follow the dispersion and be rewritten for each location in order to apply to the fragments stored in it. For instance, the query clause `SELECT x WHERE x = 269` over a 16-bit column `x` dispersed into two 8-bit subcolumns becomes `WHERE x = 1` for the high-bit location and `WHERE x = 13` for the low-bit representation, as shown in Fig. 5. Subsequently, the results of these two queries, which can be carried out in parallel without the risk of conflicts, must be reduced again on the trusted client with a `JOIN` semantics to filter out partial matches. For relations across columns, the reduction records the position (row) of the value. The set of positions can then be used as filter for the subsequent iterative processing, for instance in a query of the form `SELECT y WHERE x = 269` which then becomes `SELECT y WHERE @pos = 99` if the requested value of x can be found at this position. This method minimises cross-pollution of knowledge across cloud providers and thus maintains a high degree of confidentiality. However, it induces the need for iterative map-carry-reduce operations which involve the trusted client as intermediary.

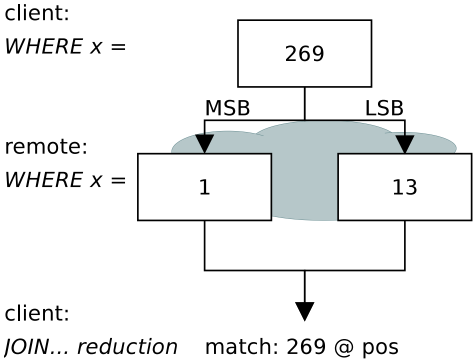

Fig. 5. Rewriting queries for each sub-column store

Stealth databases need to consider all processing primitives, including search, aggregates, predicates, comparisons and arithmetic term evaluations, to be of relevance to application developers despite the obvious restrictions due to the challenges arising from the use of specific coding and distribution schemes [12], [13]. The operators `SUM(x)` and `AVG(x)` are perfectly parallelisable. The query `SELECT AVG(ROUND(x)+1)` gives an example for a more complex query which determines the average value of a number of additions of one plus the rounded representation of each entry in the column `x`. It can still be parallelised by exploiting the split on the decimal fraction position.

Table I explains in detail how aggregate, predicate and comparison operators in the structured query language (SQL) work on dispersed data fragments. Parallel map-reduce (MR) and iterative map-carry-reduce (MCR) are possible execution modes, and homomorphic, order-preserving and searchable encryption (HE, OPE and SE, respectively) are prerequisites to enable the mode. An asterisk (*) means that an arbitrary encryption scheme will work whereas a dash (-) means that no encryption can be used for remote processing. A dash in the mode column likewise means that no dispersed remote execution is possible at all due to missing mathematical concepts for function composition. The table further informs about the possibility to achieve controlled degradation through either approximate results or at least ambiguous but small approximation sets in the case of service failures assuming the higher-bit services are still available. All operators work on `INT` and `REAL` columns except for `ROUND` which is useful for a fixed-point representation of `REAL` and `UPPER/LOWER/LENGTH` which work on `TEXT` strings. As the table reveals, more research will be necessary to allow for additional stealth processing functions. Recent research on distributed computing models suggests more generalised map-filter-sort-aggregate-reduce operator pipelines for big data processing [14] which can be allocated dynamically by consulting the operator table and therefore becomes a suitable implementation choice.

TABLE I
SQL OPERATORS WITH MAP-(CARRY-)REDUCE IMPLEMENTATIONS

Operator	Encryption	Mode	Degradation
MIN,MAX	OPE	MCR	approx-set
AVG	HE	MR	approximation
MEDIAN	OPE	MCR	approx-set
SUM	HE	MR	approximation
COUNT	*	MR	(never)
ROUND/FLOOR/CEIL	-	-	-
ABS	-	-	-
SIN/COS/TAN	-	-	-
SQUARE/SQRT	-	MCR	-
UPPER/LOWER	Caesar	MCR	-
LENGTH	Caesar,OTP	MR	(never)
=,<>	*	MR	approx-set
<,>	OPE	MCR	approx-set
LIKE	SE	MR	-

Implementations typically employ optimisations through caching of pre-calculated aggregates to deliver some of the results. This is especially true for incrementally updateable aggregates, i.e. `COUNT,MIN,MAX,AVG,SUM`.

Any encryption must be performed after the splitting because otherwise the fragments may end up containing values which have no representation in the cryptospace and thus become invalid for arithmetic operations. The queries are rewritten to contain the encrypted values, for instance `WHERE x = 349235`, and matched on the cloud side against the already encrypted values without conveying knowledge about the original match value.

C. Type-Dependent Coding

Database management systems with SQL interfaces usually support more than 20 column types. Among the fundamental ones are strings, integers, floating-point numbers and boolean expressions. Depending on the type of data, the targeted coding for subsequent remote processing is important. Floating-point numbers need to be converted to integer fixed-point approximations for arithmetic calculations when dispersion or HE are applied [15]. All integer values need to be converted to their native representation before dispersion splitting and joining occurs. Furthermore, signed integers may need conversion to unsigned types due to the calculation of logarithm and other function in many encoding steps. Without the need for arithmetics, for instance when only equality comparisons are performed, using a generic string representation of all data types will also be an option.

D. Optimisation for Non-Functional Properties

Stealth databases should be adaptive towards a user's goals regarding security, performance, cost and other non-functional properties. For this purpose, stealth databases offer a per-query goal specification for optimising the query results regarding user-specified non-functional properties. With the syntax `SELECT ... OPTIMIZE FOR <goal>`, the following goals can be specified:

1) Reliability: For values stored with replication, all replicas are fetched and compared to be equal instead of just fetching a single one. In the case of mismatches, a local majority voting is performed to determine the relative, absolute or even byzantine majority. A user-specified threshold determines which majority must be achieved. Typically, byzantine majority is required in untrusted cloud environments [16].
2) Performance: For values stored with dispersion, only the most significant fragments are processed, retrieved and combined. This method yields results with limited accuracy, precision or completeness, but achieves on average a higher performance than having to wait for all fragments to arrive. When used with fixed-point numbers, the method can be effectively used for the `FLOOR()` function without additional processing overhead.

III. STEALTH DATABASE EVALUATION

In order to evaluate the proposed concepts, we first present our prototypical implementation of a novel type of database management software called StealthDB. Afterwards, we perform stealth query experiments in both a cluster and a cloud environment.

A. StealthDB Software Design and Prototype

StealthDB is a prototypical realisation of an embedded stealth database layer for heterogeneous device and cloud environments. The main idea is to make stealth processing accessible to applications using a well-known interface which resembles relational ones. It offers an SQL and a library method interface which both allow for specification of non-functional properties upon insert and select operations.

StealthDB essentially maintains locally a set of databases, a set of tables per database, and a set of columns per table. Each column is stored in a combination of resources: main memory, files and remote storage and compute services (Fig. 6). Whenever a cloud possesses compute capabilities, processing can be offloaded to it through a deployable *StealthDB-Cloud* service. Without this capability, StealthDB gracefully falls back to local processing by fetching all effected remote columns from the storage services unless they are fully replicated locally.

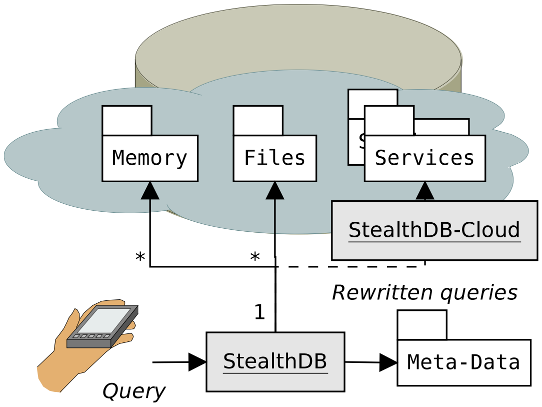

Fig. 6. StealthDB software architecture

The implementation of StealthDB is based on Python3 with the Pyro RPC and WSGI/HTTP frameworks and the Dispersed Algorithms library [5] as well as further libraries for homomorphic, order-preserving and searchable data encryption. The StealthDB-Cloud service containing the remote operators can run as standalone daemon in a VM/container or as part of a web stack, for instance in a PaaS environment such as Google App Engine, CloudFoundry or OpenShift. StealthDB offers a command-line interface similar to popular RDBMS but can also be integrated into applications as a library through a single Python module called `stealthdb.py`. Listing 1 shows a sample session when using StealthDB interactively for medical data management in a cloud environment. All cloud services are assumed to be registered in a global or per-cluster RPC naming service to avoid the direct use of hostnames. The encryption and dispersion happens transparently to the user without the need to require special SQL extensions apart from the initial specification.

Listing 1. Secure storage and retrieval of a medical record

```
USE CLOUDS ’cloud://ec2’ AND ’cloud://ostack-
  private’ WITH ’encryption,dispersion’;
DROP TABLE IF EXISTS medrecord;
CREATE TABLE medrecord (id INT, personname
  TEXT, weight REAL);
INSERT INTO medrecord (id, personname, weight)
  VALUES (23, "Alice", 63.0);
SELECT * FROM medrecord WHERE
  personname = "Alice";
SELECT AVG(weight) FROM medrecord;
```

In the sample, a triple record is stored across two cloud providers without redundancy. All three values are bit-split into two fragments according to their data types. All six fragments are encrypted into their homomorphic representation. The keypair is persisted locally and the public keys are stored along with the data in anonymised files by the two instances of *StealthDB-Cloud* workers or plain storage services. Then, both calculation and search are performed over the dispersed and encrypted data according to the algorithms for dispersed data [5].

When used in a cluster, the Pyro naming service can be queried to get a list of all registered workers. This simplifies the syntax towards `USE CLOUDS ’auto’` for large-scale clusters. Furthermore, to support long-term service evolution, new storage and compute services as well as deprecated services can be signalled to StealthDB by the naming service. The migration support, described below, ensures that the distribution of data can be progressively adapted to the evolved set of services.

If the insertion of data fails due to the non-availability of a node or a write error to the disk, the failure handler can be configured to store the data in memory within the trusted StealthDB layer temporarily and retry the write operation later. This feature is useful to keep the application logic lean in the presence of frequent network connection issues which are common with mobile applications. This feature is also used by mass inserts encapsulated into a transaction context.

For encrypted values, a key size check is performed for cases where keys cannot be concatenated and data values must not be larger than the keys. This protects against unwanted information leakage and data corruption.

StealthDB supports migrations of columns between different storage areas or coding and distribution schemes. Again, the trusted client is used as intermediate during the migration transaction. Following the example given before, Listing 2 shows the effects of migrating the weight column out of the cloud onto fully replicated local disks. The listing shows how StealthDB's debugging messages, which can enabled on demand, help to understand the background activities of the database system.

Listing 2. Migration of a medical record

```
ALTER TABLE medrecord ALTER COLUMN weight USE
  CLOUDS ’file:///securefolder’ AND ’file:///
  secureshare’ WITH ’encryption,replication’;
...
(DEBUG messages follow)
(migration:fetch)
(decryption:privkey=PrivateKey(l=19..,m=92..),
    pubkey=PublicKey(n=19..,nsq=39..,g=19..),
    entry=32..)
(decrypt-safe)
(decryption:result=13091)
(decryption:privkey=PrivateKey(l=19..,m=92..),
    pubkey=PublicKey(n=19..,nsq=39..,g=19..),
    entry=24..)
(decrypt-safe)
(decryption:result=25568)
(dispersion:decoded=63.0)
(migration:re-configure columns)
(migration:store)
(encryption:conv=909323824,bytes=b’63.0’)
(encryption:encrypt-safe)
(encryption:privkey=PrivateKey(l=26..,m=61..),
    pubkey=PublicKey(n=26..,nsq=70..,g=26..),
    data=21..)
```

The StealthDB software is publicly available to interested users[1]. A systematic test suite which runs a number of SQL statements over all possible combinations of data coding,

[1] StealthDB website: http://lab.nubisave.org/stealthdb/

distribution and storage location is also available to evaluate the fitness of the system for any application scenario.

B. Performance Evaluation

To observe the behaviour and the performance characteristics of StealthDB, we designed an experiment called the *million movie management* benchmark. The benchmark is kept very simple on purpose. It generates one million artificial movie names, inserts them into a text column in conjunction with a unique numeric identifier column, and selects all entries both ordered and unordered. Furthermore, it selects the sum of all identifiers which is a constant $\frac{n(n+1)}{2}$ for $n = 1000000$ equal to 500000500000. Whenever adequate, we determined the results for different CPU frequencies. Three computing environments with different characteristics were prepared. We tested StealthDB on a notebook, on a resource-limited Raspberry Pi cluster and in a 'big iron' private research cloud.

The benchmark was first executed on a notebook with an quad-core Intel Core i7 M620 CPU with a maximum frequency of 2.67 GHz and 6 GiB of memory running Debian 7.5, the Linux kernel 3.2.0 and Python 3.2.3. StealthDB was configured to use a single in-memory storage area. Table II shows both the internal function runtime, retrieved with `EXPLAIN ANALYZE` and `MODE quiet`, and the overall runtime, including all input and output messages on the screen and the local table structure storage, for each step of the benchmark. All measurements were performed five times to produce stable mean values. It should be noted that the slow import performance caused by a million `INSERT`s can be solved trivially with additional software engineering by adding a mass-import interface similar to for instance MySQL's bulk data loading. We consider the completeness of the database system to be out of scope for our research and focus rather on the stealth processing capabilities.

TABLE II
MILLION MOVIES MANAGEMENT RESULTS ON THE NOTEBOOK

Benchmark step	Backend	Runtime	Overall
INSERT	memory	-	245.30s
INSERT	file	-	400.17s
SELECT	memory	0.09s	1.54s
SELECT	file	0.51s	0.65s
SELECT ORDER BY	memory	2.09s	3.56s
SELECT ORDER BY	file	2.49s	2.80s
SELECT SUM(id)	memory	1.09s	2.54s
SELECT SUM(id)	file	1.47s	1.57s

In order to determine the performance overhead of homomorphic encryption, the in-memory benchmark has been repeated with encryption enabled. Table III contains the results. An interesting observation is that while the overhead is extraordinarily high in all cases, there are large deviations between the overheads which hint at separate optimisation potentials.

TABLE III
MILLION MOVIES MANAGEMENT WITH ENCRYPTION

Benchmark step	Backend	Runtime	Overhead
INSERT	memory:crypt	65685.00s	266.7
SELECT	memory:crypt	114.12s	1267.0
SELECT ORDER BY	memory:crypt	116.12s	54.6
SELECT SUM(id)	memory:crypt	110.70s	101.6

To perform the experiments in a cluster environment, we used Bobino, an 8-node Raspberry Pi cluster engineered at the Free University of Bolzano as smaller cousin of the 40-node Bobo cluster [17]. Bobino contains a gateway node which connects the internal Ethernet topology to the outside world, a brain node with custom web-based software to coordinate the allocation of all nodes, a recovery node for fault situations, and a custom-built power distribution. All nodes are Pi Model B with a 700 MHz ARMv6 CPU which can be safely overclocked to a turbo mode of 1000 MHz, 512 MiB of main memory (485 MiB free for applications) and a 100 MBit/s Ethernet port. The operating system, which boots from 4 GB SD cards, has been updated to match closely the specifications of the notebook. Bobino runs Raspbian (Debian) 7.6 with Linux kernel 3.10.25 and Python 3.2.3. We installed StealthDB workers (`stealthdb-cloud`) on each of the nodes and the system frontend (`stealthdb`) on the gateway node. Fig. 7 shows a photo of the cluster while it runs StealthDB at the default frequency of 700 MHz.

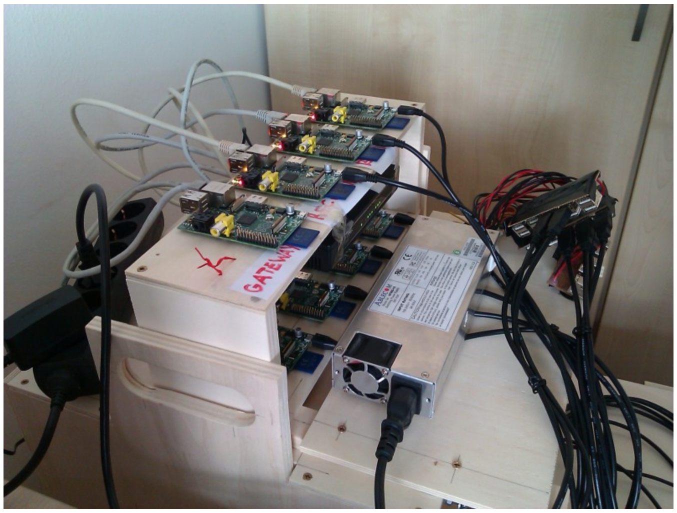

Fig. 7. Bobino, the Raspberry Pi cluster used in our experiments

Table IV summarises the performance results when running StealthDB either with a locally dispersed file-backed database or with dispersion across three nodes using RPC access to the storage and compute cloud interfaces of the nodes.

StealthDB has furthermore been installed on a cloud server system provided by the Hasso Plattner Institute's Future SOC Lab as a remote service. Compared to the other two test environments, no part of the overall server system has been under our direct control. The hardware consists of four out of 32 HP Converged Cloud blades, each with 2 Intel Xeon E5-2620 CPUs (à 6 cores/12 threads) running at 2.0 GHz and 64

TABLE IV
MILLION MOVIES MANAGEMENT RESULTS ON THE BOBINO CLUSTER

Benchmark step	Backend	Runtime
INSERT	file	19999.00s
INSERT	file:disp	30749.00s
INSERT	cloud:disp	262391.00s
SELECT	file	21.91s
SELECT	file:disp	376.30s
SELECT	cloud:disp	428.33s
SELECT ORDER BY	file	30.39s
SELECT ORDER BY	file:disp	389.22s
SELECT ORDER BY	cloud:disp	428.90s
SELECT SUM(id)	file	83.81s
SELECT SUM(id)	file:disp	490.28s
SELECT SUM(id)	cloud:disp	443.23s

GiB of memory and an NFS-connected 3PAR disk array with 3 TB capacity. The blades are interconnected by 10G Ethernet. The system runs Ubuntu 14.04 with Linux kernel 3.13.0 and Python 3.4.0.

The results of running StealthDB in the cloud are consolidated in Table V. Again, the results compare all-local dispersed files with dispersed cloud access from one blade to the three other ones. Additionally, homomorphic encryption of fragments in the cloud is used for higher protection of their sensitive contents. The principal observations are that (1) transmitting RPC messages massively decreases the insertion performance but is otherwise competitive against the local disk speed, and (2) the parallel dispersed processing over encrypted numbers speeds up the execution considerably compared with the local decryption process.

TABLE V
MILLION MOVIES MANAGEMENT RESULTS IN THE CONVERGED CLOUD

Benchmark step	Backend	Runtime
INSERT	file:disp	815.30s
INSERT	cloud:disp	17972.00s
INSERT	cloud:disp:crypt	224877.00s
SELECT	file:disp	11.99s
SELECT	cloud:disp	13.54s
SELECT	cloud:disp:crypt	444.68s
SELECT ORDER BY	file:disp	12.82s
SELECT ORDER BY	cloud:disp	14.11s
SELECT ORDER BY	cloud:disp:crypt	444.50s
SELECT SUM(id)	file:disp	14.51s
SELECT SUM(id)	cloud:disp	14.92s
SELECT SUM(id)	cloud:disp:crypt	80.67s

C. Comparison and Limitations

Insertions and basic range-aggregate queries have been implemented in StealthDB which with this functionality take less than 2000 lines of Python code. An exploration and implementation of range-aggregate deletions and updates is currently missing. Furthermore, while basic fault handling and transaction semantics are available, the database system is not guaranteeing ACID properties.

In order to allow for a comparison with popular database management systems, additional experiments have been performed on the notebook system running at 2.7 GHz. Table VI summarises the performance values when using the embedded SQlite database system, version 3.7.13, for in-memory and file-backed operations. The INSERTs have been executed in a transaction context as otherwise they would have taken up to 30 hours to execute with full ACID guarantees. The runtime for the insertion has been measured externally whereas SQlite's internal timer has been used for the SELECT statements. An interesting outcome is that both backends perform equally well with the anomaly of in-memory insertion taking longer than its file-backed counterpart. As opposed to StealthDB, SQlite does not persist in-memory data across sessions. While most metrics are faster as expected, the sorted selection is slower in SQlite. The reason for this behaviour may be caused by lexical rather than binary comparisons and needs further examinations.

TABLE VI
MILLION MOVIES MANAGEMENT RESULTS WITH SQLITE

Benchmark step	Backend	Runtime
INSERT	memory	12.27s
INSERT	file	10.83s
SELECT	memory	0.24s
SELECT	file	0.26s
SELECT ORDER BY	memory	6.60s
SELECT ORDER BY	file	6.75s
SELECT SUM(id)	memory	0.12s
SELECT SUM(id)	file	0.13s

There is a lot of optimisation potential in StealthDB by adding native methods (implemented in C), skipping sub-column aggregation if there is only one sub-column, and using persistent file handles and proper binary file formats. Fig. 8 visualises the overheads in the current StealthDB implementation when traversing different paths from plain storage in memory with local processing to distributed processing over encrypted dispersed data fragments in the cloud. The overheads prevent the system from many practical use cases where performance matters. Nevertheless, even in its current initial state, StealthDB shows potential for cases in which confidentiality, availability and other service quality properties matter. We expect production-level database systems to pick up sub-column and dispersed processing concepts in the near future.

D. Applications and Trade-Offs

Despite the mentioned limitations, we have been able to test StealthDB under load and produce applications which embed the database layer to manage sensitive data in the cloud. Among them, an e-sports tracker for managing sensor data from personal sports activities data, a travel database to record trips to places around the world, an enterprise asset management database and a secure distributed searchable document management portal have been implemented. Often, the volume of the data to be managed is small compared to

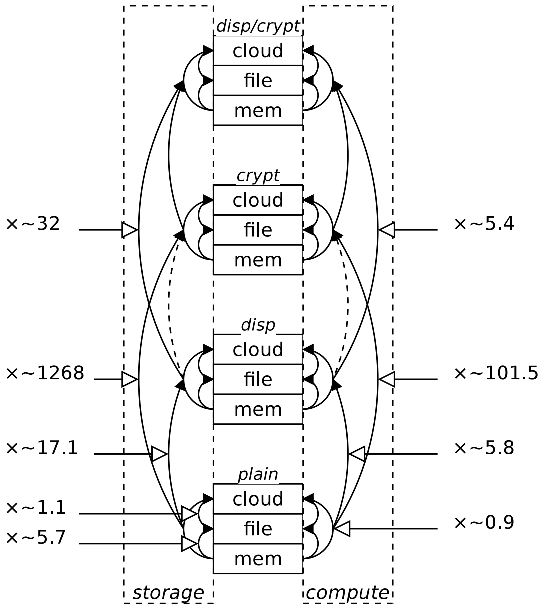

Fig. 8. Overhead factors comparison figure

its value. For such use cases, stealth database designs offer compelling benefits.

For the users, it is important to understand the significance of trade-offs. Fig. 9 demonstrates that there is a sweet spot, or pivot point, concerning the performance loss due to the stealth processing and the performance gain due to less data transmission for an aggregate query of the form `SELECT AVG(x)`. The query is once processed locally and once over an 1GBit/s Ethernet connection.

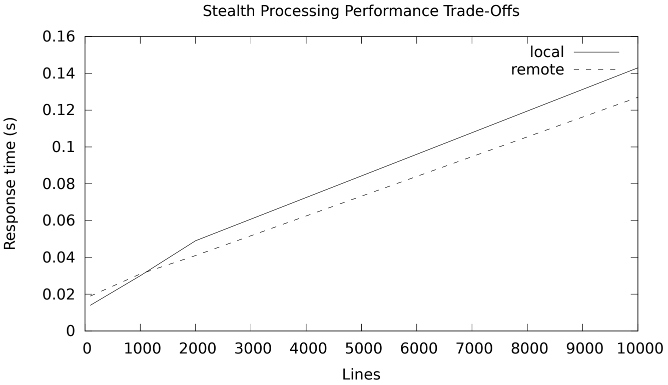

Fig. 9. Performance trade-off for an aggregate query

IV. Related Work

A number of database systems has been proposed and implemented to ensure combinations of availability, confidentiality and integrity of sensitive data and queries. Privacy Integrated Queries (PINQ) is a data management platform which applies differential privacy to its results [18]. This technique offers a trade-off between result accuracy and identity revelation. PINQ is however not distributing its data across services. TrustedDB [13] uses a hardware design to ensure a tamper-proof isolation of server parts of a database system which makes it suitable for outsourcing into the public cloud. The system guarantees data confidentiality and privacy. The main claim of TrustedDB is that the initial cost for the hardware will be recovered quickly by saving computational cost for queries over encrypted data. CryptDB is running atop a MySQL/PostgreSQL server with user-defined function extensions which encrypts its data with query-aware schemes [19]. It protects against intruders and curious administrators and has a low overhead but fails at more complex SQL queries, such as the majority of the TPC-H reference set. Monomi extends CryptDB by evaluating the problematic query parts on the client instead. This design is similar to StealthDB's although it does not represent a map-reduce pattern. Previous approaches with the same idea have been available for some time, although most of them assume a centralised query execution over holistic (encrypted or not) values [20]. Relational Cloud is a proposal to achieve a more complete protection against unavailability by adding full replication [21]. This approach requires high and potentially expensive storage capacities. In scientific computing, replication and caching of non-confidential data is commonplace to increase the processing performance [22]. In contrast, StealthDB uses partial replication through bitsplit coding to reduce the required capacity at the expense of query performance. Hence, the four properties of confidentiality, availability, capacity and performance form a rectangle in which only a triangle of three out of four parameters can be optimised for with current systems. In Fig. 10, the optimisation potential of secure cloud database systems towards more general user-requested properties, which include security protection goals, is visually compared.

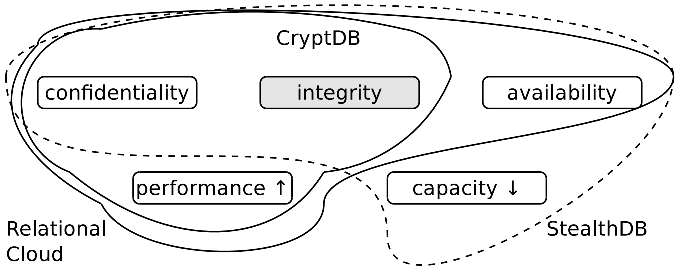

Fig. 10. Related approaches according to their support for non-functional properties

Beyond secure storage and processing of data, sharing the data among multiple users is also a concern for the design of cloud database systems. Self-controlling objects have recently been proposed for this purpose to allow for user collaboration over confidential databases [23]. Another complementary functionality for stealth layers is the check of data possession at random time intervals. Various proof-of-possession (PoP) and proof-of-retrievability (PoR) schemes have been designed with a zero or close-to-zero knowledge remaining on the client. These schemes still need to be evaluated in the context of secure databases [24].

V. Summary and Future Work

Stealth databases represent a novel distributed data management design targeted at maintaining data security in untrusted environments with low capacity but rather high performance overheads. Our prototype StealthDB, while currently only implementing a subset of SQL, allows for a thorough evaluation of stealth database concepts. The results of our experiments show that for smaller workloads such as sensitive private data, the system performs reasonably well. In the future, we intend to tackle the performance issues with optimised calculation modules. Furthermore, we want to explore novel algorithms over bitsplit codes as well as applications of dispersed processing to other coding schemes such as systematic erasure codes. Finally, we want to increase the practical utility by designing native cloud applications which embed StealthDB for their data management tasks.

Acknowledgements

This work has been partially funded by the German Research Foundation (DFG) under project agreements SCHI 402/11-1. We would like to thank the Free University of Bolzano, Italy, for designing the Bobo Raspberry-π cluster and making it available to the community[2]. Furthermore, we would like to thank the Hasso Plattner Institute's Future SOC Lab in Potsdam, Germany, for providing access to managed state-of-the-art cloud infrastructure[3].

References

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

[2] J. Spillner and A. Schill, "Towards Dispersed Cloud Computing," in *2nd IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, Chișinău, Moldova, May 2014, pp. 175–179.

[3] E. Filiol, "Formal Model Proposal for (Malware) Program Stealth," in *Virus Bulletin Conference (VB)*, Vienna, Austria, September 2007.

[4] S. Pearson and A. Benameur, "Privacy, Security and Trust Issues Arising from Cloud Computing," in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Indianapolis, Indiana, USA, November 2010, pp. 693–702.

[5] J. Spillner and A. Schill, "Algorithms for Dispersed Processing," in *1st International Workshop on Advances in Cloud Computing Legislation, Accountability, Security and Privacy (CLASP)*, London, UK, December 2014.

[6] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology - 17th Annual IACR Eurocrypt Conference (EUROCRYPT)*, ser. LNCS, vol. 1592, Prague, Czech Republic, May 1999, pp. 223–238.

[7] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order Preserving Encryption for Numeric Data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, Paris, France, June 2004, pp. 563–574.

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-Preserving Symmetric Encryption," in *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, ser. LNCS, vol. 5479, Cologne, Germany, April 2009, pp. 224–241.

[9] V. Božović, D. Socek, R. Steinwandt, and V. I. Villányi, "Multi-authority attribute based encryption with honest-but-curious central authority," *International Journal of Computer Mathematics*, vol. 89, no. 3, pp. 268–283, June 2010.

[10] R. Hans, U. Lampe, M. Pauly, and R. Steinmetz, "Cost-Efficient Capacitation of Cloud Data Centers for QoS-Aware Multimedia Service Provision," in *Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER)*, Barcelona, Spain, April 2014, pp. 158–163.

[11] S. Shirinbab, L. Lundberg, and D. Erman, "Performance Evaluation of Distributed Storage Systems for Cloud Computing," *International Journal of Computers and their Applications (IJCA)*, vol. 20, no. 4, pp. 195–207, December 2013.

[12] B. K. Samanthula, W. Jiang, and E. Bertino, "Privacy-Preserving Complex Query Evaluation over Semantically Secure Encrypted Data," in *19th European Symposium on Research in Computer Security (ESORICS)*, ser. LNCS, vol. 8712, Wrocław, Poland, September 2014, pp. 400–418.

[13] S. Bajaj and R. Sion, "TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 26, no. 3, pp. 752–765, February 2014.

[14] J. Urbani, A. Margara, C. Jacobs, S. Voulgaris, and H. Bal, "AJIRA: A Lightweight Distributed Middleware for MapReduce and Stream Processing," in *34th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Madrid, Spain, July 2014, pp. 545–554.

[15] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder, "Secure Computations on Non-Integer Values," in *2nd IEEE International Workshop on Information Forensics and Security (WIFS)*, Seattle, Washington, USA, December 2010, pp. 1–6.

[16] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and Secure Storage in a Cloud-of-Clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, November 2013.

[17] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily, and S. Bugoloni, "Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment," in *UsiNg and building ClOud Testbeds (UNICO) workshop at the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 2, December 2013, pp. 170–175, Bristol, United Kingdom.

[18] F. McSherry, "Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis," *Commun. ACM*, vol. 53, no. 9, pp. 89–97, September 2010.

[19] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing Analytical Queries over Encrypted Data," in *Proceedings of the 39th International Conference on Very Large Data Bases (VLDB)*, ser. VLDB Endowment, vol. 6, no. 5, Riva del Garda, Italy, August 2013, pp. 1–12.

[20] H. Hacıgümüş, B. Iyer, and S. Mehrotra, "Query Optimization in Encrypted Database Systems," in *10th International Conference on Database Systems for Advanced Applications (DASFAA)*, ser. Lecture Notes in Computer Science, vol. 3453, Beijing, China, April 2005, pp. 43–55.

[21] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational Cloud: A Database-as-a-Service for the Cloud," in *Fifth Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, California, USA, January 2011, pp. 235–240.

[22] D. T. Nukarapu, B. Tang, L. Wang, and S. Lu, "Data Replication in Data Intensive Scientific Applications with Performance Guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1299–1306, August 2011.

[23] D. Thilakanathan, R. Calvo, S. Chen, and S. Nepal, "Secure and Controlled Sharing of Data in Distributed Computing," in *16th IEEE International Conference on Computational Science and Engineering (CSE)*, Sydney, Australia, December 2013, pp. 825–832.

[24] N. Kaaniche, E. E. Moustaine, and M. Laurent, "A Novel Zero-Knowledge Scheme for Proof of Data Possession in Cloud Storage Applications," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Chicago, Illinois, USA, May 2014, pp. 522–531.

[2] Bobo cluster website: http://megarpi.inf.unibz.it/

[3] HPI Future SOC Lab website: http://hpi.de/en/research/future-soc-lab.html