WWHow! Freeing Data Storage from Cages

Alekh Jindal⋆ Jorge-Arnulfo Quiané-Ruiz⋄,∓ Jens Dittrich⋆

⋆Information Systems Group, Saarland University
http://infosys.cs.uni-saarland.de

⋄QCRI, Qatar Foundation
http://www.qcri.qa

Abstract. Efficient data storage is a key component of data managing systems to achieve good performance. However, currently data storage is either heavily constrained by static decisions (e.g. fixed data stores in DBMSs) or left to be tuned and configured by users (e.g. manual data backup in File Systems). In this paper, we take a holistic view of data storage and envision a virtual storage layer. Our virtual storage layer provides a unified storage framework for several use-cases including personal, enterprise, and cloud storage.

1. INTRODUCTION

To better understand the problem, let us first see some of the typical data storage scenarios and analyze what they have in common.

1.1 Current Approaches to Data Storage

We start from the simplest use case of a personal user using a File System Storage, right up to an enterprise user using Cloud Storage.
Use Case 1: File System. Consider a researcher, Alice, having two laptops (one personal, one from her university). Alice stores different types of files at different locations. For example, she stores movies on her personal laptop and grant proposals on her university laptop. Alice also maintains regular copies of her data on external devices, in case one of her laptop crashes. For instance, she maintains copies of grant proposals on hard-drives and recent conference talks on USB sticks. Finally, Alice also changes the data format (e.g. compressed) of her files in order to save storage space.
Use Case 2: RAID. Consider a University IT department using RAID storage servers to store its data. The RAID system automatically stores parity information on different disks and stripes them to allow for one or more disk failures. The server admin does not need to worry about the reliability of data. She just needs to choose the RAID level for the first time. However, changing the RAID level is often a complex task in practice.
Use Case 3: Relational DBMS. Consider a car manufacturer using an RDBMS for managing its inventory and sales. The manufacturer provides the schema definition as well as the data to store. The RDBMS takes care of physical data storage, recovery, and data layouts (e.g. row store in IBM DB2). For an expert DBA, the RDBMS provides several additional data tunings knobs such as replication, indexing, partitioning, and storage locations.
Use Case 4: Cloud. Consider a web analytics start-up company using Cloud storage to store large volumes of web logs. The start-up company needs to pick a cloud provider for its data. However, it does not need to worry about the data placement. The Cloud Storage automatically replicates and distributes data over several storage locations in order to guarantee the availability of their data. Furthermore, the start-up company does not care about adding new storage locations: the Cloud Storage does so automatically. Additionally, the company may chose to store the data in multiple data layouts [10] or indexes [5].

All of these use-cases are very different, right? No! We argue that they are all facets of the same *single* data storage problem. We observe that in each of the four Use Cases, the users are implicitly or explicitly answering the following three key questions: (1) *What* to store? (i.e. which parts to store from a dataset or file and how many times), (2) *Where* to store? (i.e. on which devices or locations), and (3) *How* to store? (i.e. in which layout or format). Table 1 categorizes these storage decisions in the four Use Cases and labels them according to whether they use fixed (■), flexible & manual (■), or flexible & automatic (■) storage decisions.
What to store? The second column of Table 1 shows the *what* decisions in the four Use Cases. In Use Case 1 (UC1 for short), the file system requires Alice to make manual choices of which *master* copies as well as the *backup* copies of data to store. In UC2, RAID automatically creates data replicas or parity, depending on the RAID level. In UC3, an RDBMS typically stores the data as well as the recovery log. A DBA has further control to create indexes and materialized views to speed up query execution. Similarly, Fractured Mirrors makes a fixed decision of storing exactly two copies of the data. Finally, in UC4, Cloud Storage is flexible to create one or more data replicas for availability.
Where to store? The third column of Table 1 shows the *where* decisions in the four Use Cases. In UC1, Alice has to manually decide where to store each type of data, e.g. movies on her personal laptop and grant proposals on her university laptop. Going further, RAID (UC2) makes fixed decisions (based on the RAID level) to store data on a set of storage devices, which can be added or removed manually. Similarly, an RDBMS (UC3) allows users to define *table spaces*, typically residing on different storage locations, for their application. However, Fractured Mirrors stores the data on two (fixed) different machines. Cloud Storage (UC4), on the other hand, is fully flexible on where to store the data. A cloud provider can change data storage locations and even provision additional physical machines automatically.
How to store? The fourth column of Table 1 shows the *how* decisions in the four Use Cases. Again, users in UC1 must manually

∓Work done at Saarland University.

6th Biennial Conference on Innovative Data Systems Research (CIDR '13) January 6-9, 2013, Asilomar, California, USA.

Storage use case		*What* to store?	*Where* to store?	*How* to store?
UC 1: File System		master copy of data, periodic backup copy	movies on personal laptop, grant proposals on university laptop	some files in original format, some files compressed
UC 2: RAID		master copy of data, one or more replicas based on RAID level	multiple storage devices	unchanged data layouts
UC 3:	Relational DBMS	master copy of data, recovery log, intermediate results (all or partially)	single machine (typically)	row or column layout, compression for some tables
	Relational DBMS + Fractured Mirrors	two copies of data	two machines	one copy of data in row layout, second copy of data in column layout
UC 4: Cloud		master copy of data, one or more replicas based on provider setting	suitable cloud provider multiple virtual machines	unchanged data layouts

Table 1: *What*, *Where*, and *How* flexibility for Use Cases 1–4. (■ unchangeable, ■ flexible & manual, and ■ flexible & automatic)

decide how to store each of the data files (e.g. original format or compressed). Similarly, RAID (UC2), does not care about the data layouts; users must decide upon them. In contrast, an RDBMS (UC3) stores data in a particular data layout, e.g. row layout in PostgreSQL. However, this is a fixed decision. Fractured Mirrors takes RDBMSs a step further by maintaining one copy in row and one copy in column layout. Finally, Cloud Storage (UC4), similar to UC2, is not concerned about the data layouts.

1.2 What is the Problem?

We observe that the decisions for *what* to store, *where* to store, and *how* to store are actually an important part of our everyday lives. However, these decisions are typically either unchangeable (■) or manual (■) in current data storage systems. Only for a very few scenarios these decisions are both flexible and automatic (■). For instance, picking an RDBMS product already cages several physical storage decisions (including the data layout) while providing an unmanageable number of tuning knobs for the others. Additionally, user applications cage the data storage even further, since only an application knows how to make full use of its data.

Therefore, the challenge is to design a storage layer that frees users (or data management systems) from all these storage level details and that has full flexibility to automatically take care of all data storage decisions. Users should only be concerned with their input data, their workload, their preferences, and their constraints.

2. OUR VISION OF DATA STORAGE

We envision a new brand of data storage layer, coined `WWHow!` (***W**hat*, ***W**here*, and ***How***; pronounced [wow!]). The core idea of `WWHow!` is threefold. First, to be fully flexible on *what*, *where*, and *how* to store data. Second, to push down all storage level logic to the data storage layer itself. Third, to free users from the burden of configuring a set of complex storage parameters (■). For this, `WWHow!` offers (i) *strong* data independence wherein the physical data definition (*where* and *how*) can be changed completely independently of the logical data definition (*what*), (ii) a declarative Data Storage Language (DSL) that is completely decoupled from the logical data languages (i.e. DDL/DML), and (iii) a holistic data storage optimizer to automatically adapt data to its access patterns and the underlying hardware. In fact, this is one of the main goals of `WWHow!`: to have a data storage optimizer that frees users from the details of data storage. Notice that, one can apply `WWHow!` to any layer of the memory hierarchy as well as to new hardware.

Figure 1 shows the `WWHow!` architecture. Users store their data directly in `WWHow!` using the DSL. Alternatively, users employ a data managing system, e.g. an RDBMS, which in turn uses the DSL to interact with `WWHow!`. The `WWHow!` layer translates the DSL (`WWHow!` language) statement into a physical data operation request and sends it to the `WWHow!` controller. The `WWHow!` controller inspects whether or not the data operation request could be optimized. Indeed, `WWHow!` users (end users or data managing systems) can (i) exercise full control over their data operations, i.e. full

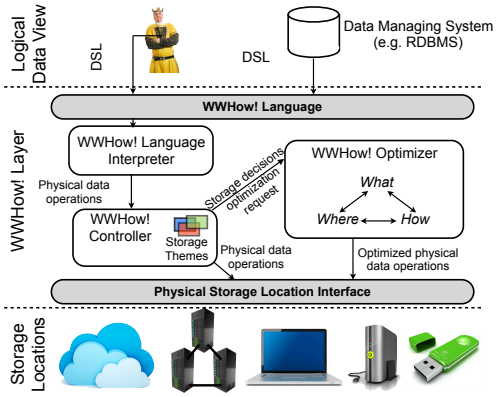

Figure 1: `WWHow!` architecture

specification of *what*, *where*, and *how* to store; (ii) provide hints in the form of storage preferences and constraints, which can be saved as *storage themes*; or (iii) completely leave the data storage decisions to `WWHow!`. In case no optimization is needed, the `WWHow!` controller sends the data operation request directly to the storage locations. Otherwise, it sends the data operation request to the `WWHow!` optimizer. The `WWHow!` optimizer tries to come up with a better data operation request, with full flexibility of *what*, *where*, and *how* to store. In the worst case, the optimizer falls back to the standard non-optimized data operation requests being used today. We discuss several examples, below in Section 3, to illustrate these new data storage scenarios. Notice that, applications can still optimize the way they access data as they can get all details on how `WWHow!` stores its data.

3. `WWHow!` NOVEL APPLICATIONS

A key feature of `WWHow!` is an easy-to-use declarative Data Storage Language, which provides complete control over what, where, and how to store. The `WWHow!` language grammar is as follows:

```
STORE | FETCH | DELETE | UPDATE | DESCRIBE url
 [WHAT {query_expression}]
 [WHERE {url}]
 [HOW {layout]
 [CONSTRAINTS {constraint}]  [PREFERENCES {preference}];
```

It is worth noticing that `WWHow!` leverages existing query languages (such as SQL) to specify which parts of data (the `WHAT` clause) from a data source (the url in the `STORE` clause) a user wants to store. `WWHow!` offers several storage applications that are either not possible or hard to do in traditional storage systems. For space constraints, we discuss only some of them below.

(1) `WWHow!` File System. Consider again Alice from UC1. Using File System Storage, she has to make several manual storage decisions (■ for UC1 in Table 1). She can still do the same using `WWHow!`. For instance, she can store her conference talks (her .pdf files twice and her .ppt files once) in encrypted format (using rsa) on a university server as follows:

```
STORE '/Users/alice/conferences/talks/*.*'
WHAT *.(pdf | ppt | key), *.pdf
```

```
WHERE infosys.uni-saarland.de/talks/
HOW encryption(rsa) FOR *;
```

However, as Alice becomes more successful, using WWHow!, she can change her data storage over time. She can simply specify her new storage requirements and let WWHow! handle the rest. For example, as she travels for various research activities, she may want to have access to her data all the time and from anywhere. WWHow! allows her to specify this *preference* using the following WWHow! language statement:

```
STORE '/Users/alice/conferences/talks/*.*'
WHAT *.(pdf | ppt | key), *.pdf
PREFERENCE Availability='high';
```

WWHow! optimizer then automatically creates redundant data copies of her data across several storage locations (e.g. personal laptop, university servers), without Alice needing to fiddle with them. Notice that, as with any automatic optimization in databases, users loose some control with the WWHow! optimizer. However, we believe that the optimizer will come up with better decisions in most of the cases. Now, assume Alice wants to make sure to not loose her data, as it gets distributed across several storage locations. In contrast to a preference, which is rather soft, this is a strict requirement for Alice. Thus Alice specifies this as a *constraint*:

```
STORE '/Users/alice/conferences/talks/*.*'
WHAT *.(pdf | ppt | key), *.pdf
CONSTRAINT FaultTolerance='high';
```

Thus, we see that in contrast to a file system, WWHow! allows for automatic decisions to evolve data storage over time ( ).

(2) WWHow! RAID. Consider again the university IT department from UC2. Using RAID Storage, the server admin not only makes manual decisions ( ), but also lives with certain fixed decisions ( ). Instead, using WWHow!, the server admin can create a reliable RAID-1 like storage wherein Cloud Storage locations can be added as the university grows in size[1]. WWHow! RAID is not just flexible in terms of storage locations but also in terms of data layouts. For example, the server admin may see high update workloads at some times (e.g. beginning of semester) and high read workloads at others (e.g. during the semester). Using WWHow!, she can transform her data to keep mirrored copies, for fast parallel reads, instead of RAID parity for reliability. WWHow! can further boost read performance by storing each mirrored copy in a different layout and, thereafter, directing a parallel read to the most appropriate layout [10]. Thus, we see that in contrast to standard RAID and Cloud Storage, WWHow! offers full control as well as automates several decisions on *what*, *where*, and *how* to store data ( ).

(3) WWHow! Relational DBMS. Consider again the car manufacturer from UC3. Using a traditional RDBMS Storage, the manufacturer is tied to several manual ( ) as well as fixed ( ) storage decisions. Instead, using a WWHow! enabled RDBMS, the manufacturer can scale its data across different data centers. However, the manufacturer may have different hardware at different (or same) data centers. This affects query processing robustness, a key goal in enterprise data management. WWHow! handles this by considering the hardware characteristics while storing data pages. Furthermore, the manufacturer must achieve certain SLAs as its business and data grows. The RDBMS may in turn translate these SLAs into data access time constraints. WWHow! can handle such constraints by automatically replicating data pages (fully or partially), adjusting page and buffer size, and adapting internal page layouts. Again, we see that WWHow! allows the manufacturer to flexibly and automatically evolve its RDBMS with the business ( ).

[1]This abstraction between logical and physical data is similar to tablespaces in Oracle; however, in contrast to tablespaces, WWHow! is not a one-to-one mapping from logical to physical data.

(4) WWHow! Cloud. Consider again the start-up from UC4. Using traditional cloud storage the company has to pick the *right* cloud provider ( ). However, with WWHow! the company can simply specify its preferences and constraints. For example, the company may have privacy concerns for its data (due to competitors or user agreements). It is more natural and easier for the company to simply provide this constraint and let WWHow! optimizer to store data with the more privacy secure cloud provider[2]. Furthermore, since the company is in the initial stages, it has limited budget for its IT department. Using WWHow! the company can fix the maximum money they want to spend on Cloud Storage. The WWHow! optimizer will automatically decide *what*, *where*, and *how* to store data on the Cloud, keeping monetary costs within the budget. Finally, the company can exploit the full flexibility of WWHow! and achieve both fault tolerance and high workload performance at the same time. To do so, the company can create three *full* redundant copies of its web logs data and create different clustered indexes for each copy (as in [5]) as follows:

```
STORE '/System/webApp/logs/uservisits.log'
 WHAT * AS replica-1, * AS replica-2, * AS replica-3
 WHERE ec2-007-23-167-120.compute-1.amazonaws.com
 HOW Idx(url) FOR replica-1,
     Idx(sourceIP) FOR replica-2,
     Idx(visitDate) FOR replica-3;
```

Thus, we see that with WWHow! even the Cloud Storage is not a taken-for-granted storage system anymore. Instead, it is fully flexible, agile, and adaptive ( ).

4. WWHow! ADVANTAGES

In the previous section we saw several examples of how WWHow! can dramatically change the data storage experience. Now below let us extract the major advantages that WWHow! offers.

(1) Flexible Storage Control. Users are free to choose which storage decisions they want to specify, while WWHow! takes care of the unspecified storage decisions.

(2) Freeing Users From Storage Decisions. Users are no longer confronted with a bunch of storage tuning knobs to obtain the best performance for their applications. However, users can still exercise partial (via preferences) or full control (via constraints) over specific data storage aspects. The examples of Section 3 clearly illustrate this flexibility.

(3) Physical Data Independence. Theoretically, DBMSs claim to provide physical data independence [7, 4, 12]. However, in practice, current DBMSs fail in doing so (see Chapter 2.3 of [7]). For example, to partition a table vertically in Oracle 11g (as well as in other databases), DBAs need to create and load a new table for each vertical partition. Then, to access multiple vertical partitions, users need to formulate join queries (or create views) over the different vertical partitions. One of the goals of WWHow! is to hide all these details from users.

(4) Flexible Decisions. Users can easily change their storage decisions at any time, without affecting their application logic. This is either impossible or very hard to realize in current data storage systems. For example, to modify vertical partitioning in ORACLE 11g, users need to create new tables and drop the old ones.

(5) No Storage Cages. Currently, data storage systems have a hard-coded data store, i.e. using a given data storage system implies using a given fixed data layout. MySQL allows developers to build and install custom storage engines. However, this approach still requires expert DBAs to design and skilled developers to develop the new storage engine. Additionally, enterprise users end up having many features replicated across different storage engines.

[2]The privacy levels could be self-declared by cloud providers, or estimated by rating agencies, or even gauged by public opinion.

WWHow! significantly departs from this approach. Using WWHow!, users can easily adapt data storage to their needs over time.

(6) Application Interoperability. Current data storage systems typically have a very strong coupling between their query processor and data store. As a result, users are tied to a query processor as soon as they choose a data store. Instead, WWHow! allows users to deploy the most suitable query processor as well as to switch the query processor at any time. Applications have only to send a logical query plan to WWHow!, such as in federated query processing. WWHow!, in turn, optimizes each received logical query plan and produces a physical query plan accordingly.

Notice that, for space constraints, we cannot discuss all the interesting aspects of WWHow! in this paper.

5. WWHow! RESEARCH AGENDA

Our vision of data storage leads to several interesting research challenges. We sketch our research agenda below.

First, in order to realize the WWHow! dream, we need to change the way we design data management systems. Data management systems must completely decouple data storage from query execution. Ideally, we should be able to store and manage data independent of the data applications. Data applications should be able to simply *sit* on top of the data. For this to happen, future data management systems must be able to push down storage level logic to a dedicated and fully flexible storage layer (such as WWHow!). This will not only improve application performance dramatically but also bring forth several novel data applications, such as those discussed in Section 3. Furthermore, future data management systems must be able to allow several applications to operate on the same data. This means that the same data can be harnessed seamlessly across several systems, without needing costly ETLs. All this requires data management system designs to have a better separation between the logical and the physical data definitions.

Second, data storage should not be treated as fixed decisions. Instead, we need to *constantly* adapt data storage to the application's needs. This is in fact one of the main goals of WWHow!, i.e. to adapt data storage to users' needs. To make the WWHow! dream a reality, we need to understand the data access/update patterns in order to make the right storage decisions. For this, we need to come up with (i) efficient techniques to monitor data access/updates, (ii) online algorithms to detect changes in access/update patterns, and (iii) prediction models for future update/access patterns. Achieving this will allow WWHow! to adapt data storage to any change in the needs of users' applications.

Third, WWHow! needs to make holistic storage decisions for *what*, *where*, and *how* to store data. While users may explicitly specify one or more of these decisions, WWHow! must automatically figure out the missing ones. Furthermore, we need to optimize for the different storage constraints and preferences specified by users. The challenge here is that users' constraints and preferences might be antagonistic and hence hard to come up with the right storage decision. For example, a user might desire to store his data on a highly available data storage (indicating a Cloud storage location) and with high data privacy (indicating a non-Cloud storage location) at the same time. This calls for both developing new techniques to create and adapting flexible physical data designs.

Finally, we live in an information age and storing data is a part of our everyday life. Therefore, we need to develop interfaces which offer simple yet efficient data storage, i.e. users should have complete control over their data and they should also find it easy to use. For WWHow!, this means we need to develop an abstract storage interface that allows for both: (i) users to have full control over its data, and (ii) applications to interoperate on the same data. The WWHow! language we presented in Section 3 is just the first step towards this direction.

6. RELATED WORK & CONCLUSION

File and personal information management systems, e.g. [6], help users to organize their data on personal computers. Novel features like Apple MobileMe and Windows Live further allow users to synchronize their data across devices. However, personal information management is still very limited in flexibility of *what*, *where*, and *how* to store data.

RAID servers are standard for recoverable data storage. However, often in practice, once the system is installed, it is very difficult for the administrator to change the RAID level. Though recent works such as [11, 2] are steps towards a more flexible *where* part in RAID, the *how* part still remains unexplored.

Database management systems provide a lot of storage tuning knobs. For instance, users can create materialized views [8] (*what* part), define tablespaces (*how* part), or turn the physical database design knobs (e.g. partitioning, indexing, cracking [9]). However, databases have a fixed data store per product, e.g. row store in PostgreSQL. RodentStore [3] provides a storage language, but it requires users to manually specify tedious storage algebra expressions. WWHow! language, on the other hand, is more user-friendly. Still, WWHow! language statements could be compiled to RodentStore storage algebra for storage optimization. Fractured Mirrors [13] makes a fixed decision of two data copies, one in row and one in column layout. However, Fractured Mirrors (as well as [10]) focusses on *how* to store the data. Instead, WWHow! offers full flexibility of *what*, *where*, and *how* to store data.

Cloud services offer scalable data storage e.g. Amazon S3 [1]. However, these services create storage cages for user data. In general, Cloud services automatically manage *where* to store the data. Nevertheless, they still leave the *what* and *how* part unanswered.

Conclusion. In this paper, we identified *what*, *where*, and *how* as three key aspects of data storage. We presented WWHow!, a holistic data storage layer that is fully flexible to decide *what*, *where*, and *how* to store data. We believe that the WWHow! layer, along with the WWHow! language, allows for many novel and exciting data storage applications, such as all-in-one personal data storage, RAID over Cloud, and replicated storage for multiple indexes/layouts.

Acknowledgments. Work partially supported by BMBF.

7. REFERENCES

[1] Amazon Web Services, aws.amazon.com.
[2] M. Balakrishnan et al. Differential RAID: Rethinking RAID for SSD Reliability. In *EuroSys*, 2010.
[3] P. Cudré-Mauroux et al. The Case for RodentStore: An Adaptive, Declarative Storage System. In *CIDR*, 2009.
[4] C. J. Date. *An Introduction to Database Systems*. Addison Wesley, 8th edition, 2004.
[5] J. Dittrich, J.-A. Quiane-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad. Only Aggressive Elephants are Fast Elephants. *PVLDB*, 5(11), 2012.
[6] J.-P. Dittrich et al. iMeMex: Escapes from the Personal Information Jungle. In *VLDB*, 2005.
[7] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 5th edition, 2007.
[8] J. Goldstein et al. Optimizing Queries Using Materialized Views: A Practical, Scalable Solution. In *SIGMOD*, 2001.
[9] S. Idreos et al. Database Cracking. In *CIDR*, 2007.
[10] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich. Trojan Data Layouts: Right Shoes for a Running Elephant. In *SOCC*, 2011.
[11] O. Ozmen et al. Workload-Aware Storage Layout for Database Systems. In *SIGMOD*, 2010.
[12] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2003.
[13] R. Ramamurthy et al. A Case for Fractured Mirrors. In *VLDB*, 2002.