The ObjectStore Database System

Charles Lamb, Gordon Landis, Jack Orenstein, Dan Weinreb
(1991)

Devyani McLaren

(slides adapted from Ricardo Pedrosa, Jian Xu)

Feb 14th 2023

CPSC 504

Who’s gonna take the biggest chunk of the pie?

A **B I G** Company has a need for DBMSs to control the **WIDEST – HUGHEST** amount **EVER** seen of applications. Two options come to the plateau: one group of developers offering the somewhat well known Relational model and another one offering an object oriented approach based on Object Store.

The **B I G** Company decides to hear what’s good and bad of this new approach before deciding in whose hands each one of its applications are going to end.

Motivation

- Impedance mismatch between application code and database code (eg, C++ and SQL)
- ObjectStore provides a uniform programmatic interface to both **persistent** and **transient** data.

Persistent = data stored in a database

Transient = data when running a program

C++

- Object-oriented (OO) programming language
  - Classes w/
    - Abstraction, encapsulation, inheritance, polymorphism
- Pointers
  - Variable the stores the memory address of an object
  - Can ‘directly’ manipulate memory (more low-level than other languages)
  - But! Can lead to a lot of memory mixup and errors because of empty memory that may not be allocated for

Goal: add persistence to C++

- Ease of learning: C++ plus a little extra.
- No translation code: persistent data is treated like transient data.
- Expressive power: general purpose language (as apposed to SQL)

- Reusability: same code can operate on persistent or transient data
- Ease of conversion: data operations are syntactically the same for persistent and transient data.
- Type checking: same static type-checking from C++ works for persistent data.
- Temporal/Spatial locality: take advantage of common access patterns.
- Fine interleaving: low overhead to allow frequent, small database operations
- Performance: do it all with good performance compared to RDBMSs

Application Interface

- Three programming interfaces: libraries for C and C++, and an extended C++ language. We focus on language extension.
- Keyword **persistent**. Used when declaring variables
- A few other keywords (**inverse_member, indexable**) for defining how objects in the DB relate.

```
main()
{
  database *db = database::open("/company/records");

  persistent<db> department* engineering_department;

  transaction::begin();

  employee *emp = new(db) employee("Fred");
  engineering_department->add_employee(emp);
  emp->salary = 1000;

  transaction::commit();
}
```

Collections

- Similar to arrays in PL’s or tables in DBMSs
- Allow performance tuning: developers specify access patterns and an appropriate data structure is chosen
- Elements may be selected from collections with queries

Relationships
(this can be skimmed or skipped as needed)

- Pairs of inverse pointers which are maintained by the system.
- One-to-one, one-to-many, and many-to-many are supported.
- Syntactically, relationships are C++ data members, however, updating causes its inverse to be updated.

Associative Queries

- Selection predicates can be applied to collections.
- Special syntax: `[: predicate :]`
- Eg.

```
employees [: salary >= 10000 :]
```

- Queries may be nested.

Approx Query example

os_Set(employee*)> & overpaid_employees=
all_employees->query(‘employee*’, “ [: salary >=
100,000:]”);

C++

os_Set(employee*)& overpaid_employees =
all_employees [: salary >= 100,000 :];

Extended C++

SELECT * ~~salary~~
FROM all_employees ??
WHERE salary >= 100,000

SQL

Accessing persistent data

- Overhead is a major concern.
- Once objects have been retrieved, subsequent references should be as fast as an ordinary pointer dereference.
- Similar goals as a virtual memory system-- use VM system in OS for solution

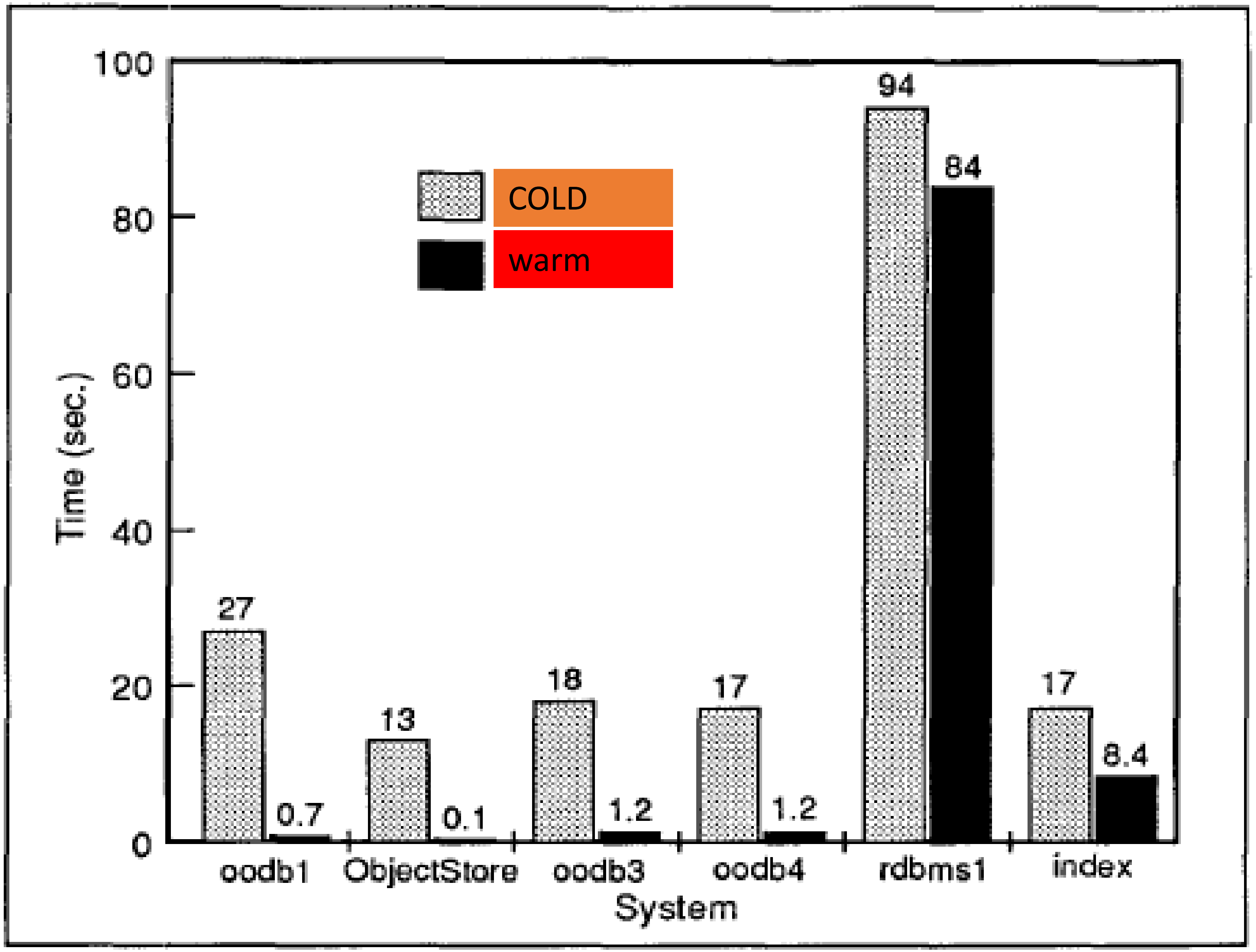
Query optimizations

Some RDBMS query optimization techniques don’t work or make sense

- Collections are not known by name
- Join optimization is less of a problem
  - paths can be viewed as precomputed joins
  - optimization is index selection
  - “true joins” are rare
- Index maintenance is more of a problem

Discussion (from Matt)

Learning from the history of “excellent” ideas that didn't make it - should we be cautious about deployment right away? How can we tell when the winds favor us? Is it luck, or is there something we can research/plan for?



100
80
60
40
20
0
Time (sec.)
COLD
warm
27
0.7
13
0.1
18
1.2
17
1.2
94
84
17
8.4
oodb1
ObjectStore
oodb3
oodb4
rdbms1
index
System

Conclusion

- Performance experiments show caching and virtual memory-mapping architecture work.
- Small case study shows productivity benefits
- ObjectStore provides
  - Ease of use
  - Persistent C++
  - Expressive power
  - High performance due to VM mapping architecture

Who's gonna take the biggest chunk of the pie?

A **B I G** Company has a need for DBMSs to control the **WIDEST – HUGEST** amount **EVER** seen of applications. Two options come to the plateau: one group of developers offering the somewhat well known Relational model and another one offering an object oriented approach based on Object Store.

The **B I G** Company decides to hear what's good and bad of this new approach before deciding in whose hands each one of its applications are going to end.

Of Objects and Databases: A decade of Turmoil

Michael J. Carey, David J. DeWitt
(1996)

Devyani McLaren

(slides adapted from Ricardo Pedrosa, Jian Xu)

Feb 14th 2023

CPSC 504

Objects and Databases. Areas of research

- Extended relational database systems.
- Persistent programming languages.
- Object-oriented database systems.
- Database system toolkits/components.

Extended Relational Database Systems

Areas of Research 1

- Allow the additional of new, user-defined abstract data types (ADTs).
  - ADTs are implemented in an external language
  - After being registered with the database, ADT’s functions can be used in queries

- Projects
  - Ingres
  - Postgres
    - Query optimizers with ADT’s properties and functions awareness
    - Support for storing and querying complex data types

Persistent Programming Languages

Areas of Research 2

- Add data persistency and atomic program execution to traditional object-oriented programming languages

- Problems addressed:
  - Impedance mismatch

Object-Oriented Database Systems

Areas of Research 3

- Combine all of the features of a modern database system with those of an object-oriented programming language, yielding an objected-oriented database (OODB) system.

- Focused on:
  - Support for querying, indexing and navigation
  - Addressing version management needs of engineering apps

Database system toolkits/components

Areas of Research 4

- Provide a DBMS that can be extended at almost any level and have additional tools that help building domain-appropriate DBMS

- Projects:
  - Exodus
    - Storage manager for objects
    - E: a persistent programming language
    - Query optimizer generator
  - Starburst
    - Clean architectural model that facilitates storage and indexing extensions
    - Rule-based extensible query subsystem

What happened?

Objects & Databases in 1996

- **System toolkits & persistent programming languages:**
  - Despite some interesting results these were a failure from a commercial POV
- **OO Database systems:**
  - Many results from the academic POV. Not expanded commercially as expected by its developers
- **Language-specific object wrappers for relational databases:**
  - New approach that appears to be important for building OO, client-side apps
- **Extended relational DBS**
  - Renamed as Object-Relational DBMS. Appears to be a settling in terms of providing objects for enterprise DB apps

The database toolkit approach problem

Causalities 💔

- Require a lot of expertise
- End up in being inflexible awkward of incomplete
- As OO and O-Relational database systems provide enough extensibility, it's not worthy to start from scratch even given a toolkit to help in the process

Why Exodus failed?

Causalities 💔

- The client/server architecture introduced an unwanted level of indirection when users tried to use EXODUS to implement their own object servers
- E programming language: Too general for skilled database implementors and too low-level for application-oriented programmers
- The query optimizer was inefficient and hard to use

Persistent Programming Languages

Causalities 💔

- No commercial implementation of such a language
- Still active as a research area in academia
- Work on this area has had a significant impact and has been transferred to OODBMS
  - Navigational programming interfaces
  - Persistent models
  - Garbage collection schemes for persistent data

What OODBM's must support

Object-Oriented Database Systems (OODBMS)

- Complex objects
- Object identity
- Encapsulation
- Inheritance & substitutability
- Late binding
- Computationally complete methods
- Extensible type system
- Persistence
- Secondary storage management
- Concurrency control
- Recovery
- Ad hoc queries

What OODBM's might support

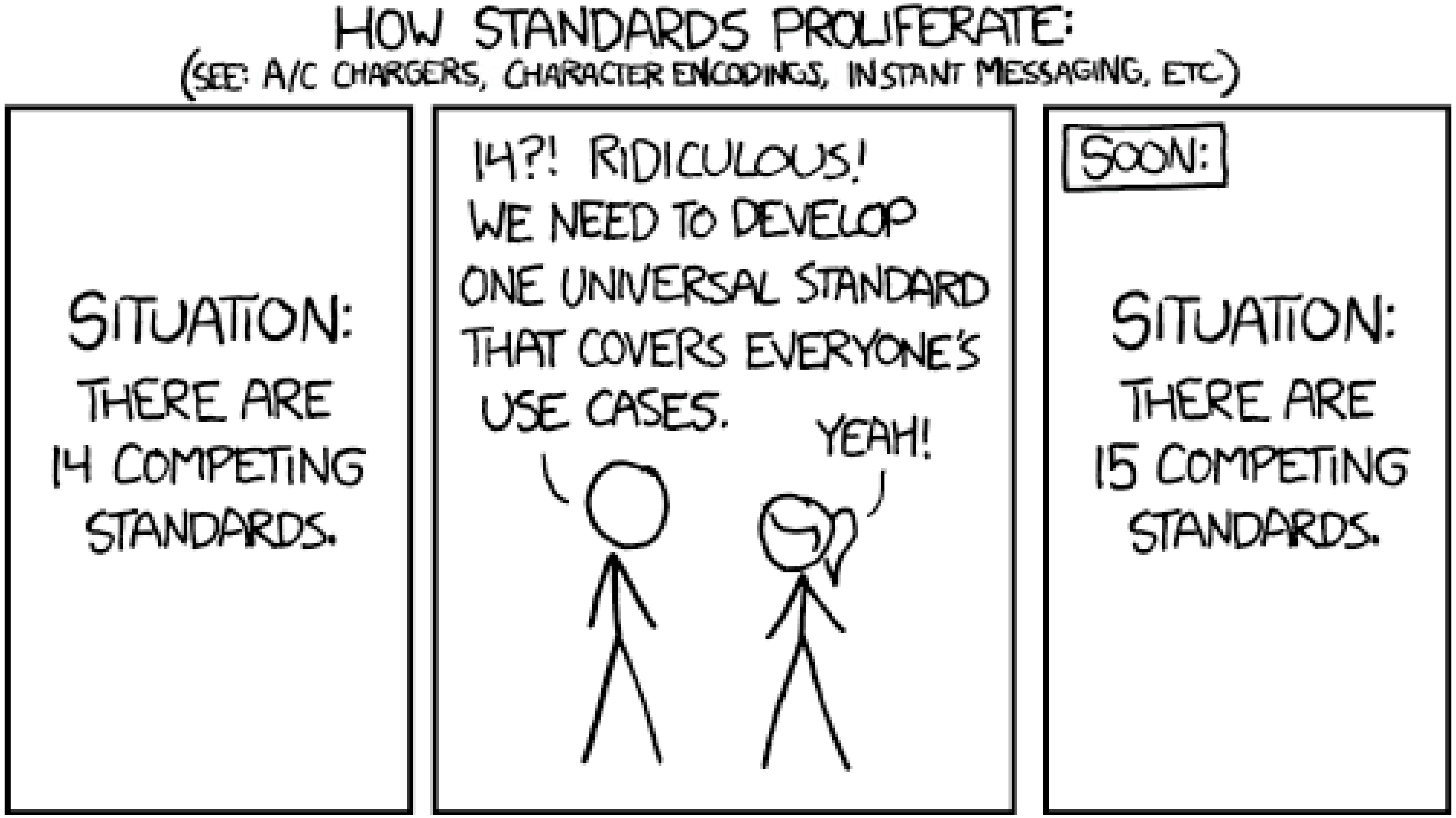
- Multiple (vs. single) inheritance
- Static (vs. dynamic) type checking)
- Distribution
- Long transactions
- Version management

What went wrong with OODMS?

Object-Oriented Database Systems (OODBMS)

- Lack of standards
- OODBMS products are behind RDMS in some terms (e.g. no views)
- Painful schema evolution
- Low availability of application development tools

Standards



https://xkcd.com/927/

Standards (from Jeffrey and Carol)

- What situations/environments lend themselves well to standardization efforts? Which ones devolve into conflict and disagreement?

Main tenets for ORDBMS (aka extended DBMS)

Object-Relational Database Systems (ORDBMS)

- Provide support for richer object structures
- Subsume RDBMS
- Be open to other subsystems (tools and multi-database middleware products)

What ORDBMS should provide?

- A rich type system, inheritance, functions, and encapsulation, optional unique ids and rules/triggers
- A high-level query-based interface, stored and virtual collections, updatable views and separation of data model and performance features

Fully integrated solution

A vision from 1996 -> 2006

- Object relational servers will provide:
  - Support OO ADTs
    - Inheritance among ADTs
    - ADT Implementation in various programming languages
  - Full OO support for row types
  - Support for middle-tier and desktop applications
  - Methods and queries will be run on cached data on servers or clients depending on where's faster

Research Challenges

A vision from 1996 -> 2006

- Server functionality and performance
- Client integration
- Parallelization
- Legacy data sources
- Standards

Many of you, but explicitly Michael

- What determines which solution “wins” out in the end, if any?

Bonus discussion (Yingfeng)

What happens to an area if it lacks commercial value but still has academic/engineering value？