The ObjectStore Database System
Charles Lamb
Gordon Landis
Jack Orenstein
Dan Weinreb

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

A BIG 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 BIG 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.

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.

Goal: add persistence to C++

- 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 or tables in DBMSs
• Allow performance tuning: developers specify access patterns and an appropriate data structure is chosen
• Similar to using collection interfaces in modern libraries (Java, C#)
• Elements may be selected from collections with queries (more on this to come).

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.
• How does this work for the library interface?

Associative Queries
• Selection predicates can be applied to collections.
• Special syntax: [: predicate :]
  Eg.
  employees [: salary >= 10000 :]
• Queries may be nested.

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:
  – Set flags so that accessing a non-fetched persistent object causes page fault.
  – Upon fault, retrieve object.
  – Subsequent access is a normal pointer dereference

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
Conclusion

- Performance experiments show caching and virtual memory-mapping architecture work.
- Small case study shows productivity benefits
- ObjectStore provides
  - Ease of use
  - Expressive power
  - Tight integration with host environment
  - High performance due to VM mapping architecture

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

A BIG Company has a need for DBMSs to control the WIDEST – HIGHEST 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 BIG 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.