The ObjectStore Database System

Paper By: Charles Lamb, Gordon Landis, Jack Orenstein, Dan Weinreb

Presented by: Sapna Bedi
Discussion by: Fong
Motivation

- Impedance mismatch between application code and database code (E.g. 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 locality: many data items will be used mostly by one user over a short span of time.
- Spatial Locality: Application uses a small (contiguous) portion of the database.
- Fine interleaving: low overhead to allow frequent, small database operations.
Discussion 1: What is more important?

- Ease of learning
- No translation code
- Expressive power
- Reusability
- Conversion
- Type checking
- Temporal locality
- Spatial locality
- Fine interleaving

1. Which goal do you think is the most and least important?
2. Why?
Application Interface

- Three programming interfaces: libraries for C and C++, and an extended C++ language. We focus on language extension.
- Keyword **persistent** - used to declare variables residing in a specific database
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
- Elements may be selected from collections with queries (more on this to come)
Collections

```c
/* file records.H */

class employee
{
  public:
    char* name;
    int salary;
};
class department
{
  public:
    os_Set(employee*) employees;

    void add_employee (employee *e)
    { employees->insert (e); }

    int works_here (employee *e)
    { return employees->contains (e); }
};
```

Figure Source: The ObjectStore Database System, By: Charles Lamb, Gordon Landis, Jack Orenstein, Dan Weinreb
Relationships

- Pairs of inverse pointers which are maintained by the system.
- One-to-one, one-to-many, and many-to-many are supported.
- Relationships are C++ data members
- Updating one relationship also causes its inverse to be updated.
Associative Queries

- Query: expression that operates one or more collections
- Selection predicates can be applied to collections.
- Special syntax: `[ : predicate : ]`
- E.g. `d -> employees
  employees [ : salary >= 10000 : ]`
- Queries may be nested
- E.g.
  `all_employees
  [:dept -> employees[ :name == `Fred` : ] : ]`
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
Accessing persistent data

- Server stores and retrieves pages of data in response to requests from clients
- Does concurrency control and recovery
- Query processing done at the client
- Pages of data are placed in client cache – allows efficient interleaving of computation and database access
Query optimizations

Some RDBMS query optimization techniques don’t 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
- Index maintenance is more of a problem
Conclusion

- Performance experiments show caching and virtual memory-mapping architecture work.
- ObjectStore provides
  - Ease of use
  - Expressive power
  - Tight integration with host environment
  - High performance due to VM mapping architecture
Discussion 2 : Relational or Object Oriented?

- Would you rather use a relational database, or Object Store?
- List applications you would use with them and why.
Thank You!