Advanced Topics in Computer Systems, CS262B
Prof Eric A. Brewer

Object-Oriented DBMS

February 24, 2004

[based on notes from Joe Hellerstein]

I. Background

Recall our Friend, The Relational Model:

- DB = {relations}
- Relation = {tuples}
- Tuple = {named fields/columns (homogeneous)}

Relational Languages

- SQL @ declarative queries (or QBE, Quel, etc.)
- C/SQL or 4GLs for applications

Other relational goodies

- Views vs. Logical vs. physical schemas (data independence)
- Triggers, authorization, constraints
- Simple algebra & query optimization
- Robust systems w/good performance
- Easily parallelizable

Q: Isn't this heaven?

A1: "A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers"
A2: E/R world, set-valued attributes, variances among entities, &SQL limitations (expressive power)

OODBMS Goals

1. Shrink the "impedance mismatch" problem for application programmers
   - DB vs. PL type systems
   - Declarative vs. procedural programming
   - Set-at-a-time vs. instance-at-a-time compilation
2. Relax data model limitations
   - Atomic values, tuples, sets, arrays, identity

- Classes w/methods & encapsulation
- Subtyping/inheritance
- Composite objects (w/sharing)
- Versions/configurations (& long xacts)

3. New language features
   - Computationally complex methods (e.g. C++)
   - Complex object "queries"
   - Integrated DBPL(s) (sometimes)
   - General focus tends to be on CAx, GIS, telecomm, cooperative/collaborative work, etc.
   - Predates "Object-Relational" systems (but coeval with Postgres)

The OODBMS Manifesto (Atkinson/Bancilhon/DeWitt/Dittrich/Maier/Zdonik, '90)

Thou shalt support:

1. Complex objects (tuples, sets, bags, arrays + constructors & ops)
2. Object identity (equal not the same as identical; sharing & updates. Plutarch's Ship of Theseus)
3. Encapsulation (ADTs/info hiding/implementation vs. interface)
4. Class/type hierarchies (inheritance, substitution for specialization)
5. Late binding (polymorphism, "virtual" classes in C++ terms)
6. Computational completeness (methods)
7. Extensibility (system & user types are the same)
8. Persistence (orthogonal to type)
9. Secondary storage (large DBs)
10. Concurrency control
11. Recovery
12. Ad hoc query facility (declarative, optimized)

Thou may support:

1. Multiple inheritance
2. Type checking (static vs. dynamic up to you)
3. Distribution (client/server)
4. Long xacts
5. Version management

Wide open:

1. Programming paradigm
2. Type systems details (base + constructors)
3. Type system fanciness (e.g. templates, etc.)

II. ObjectStore

One of the more successful vendors, both commercially and design-wise.Took C++ type system & language constructs, added databasey featuressuch as:

- Persistence for objects (at allocation)
- Bulk types (via templates)
- Relationships (i.e. OO referential integrity)
- Query expressions (simple, but optimizable)
- Fancy runtime system with DB goodies like CC&R, client/server, indexes,…

Some simple DDL examples to see data model, C++ extensions, "query language", index support.

Still in business (www.odi.com), supporting C++, Java. Started an XML product called Excelon, and now ExcelonCorp is the "parent" company, ODI the child.

III. Major Research Themes in OODBMSs

Pointers:

- Logical pointers (i.e. disk pointers) require a level of indirection (hash index over the buffer pool). The level of indirection consumes time *and* space!
- Physical pointers (i.e. memory pointers) require the level of indirection to be translated. *Pointer swizzling.* ObjectStore "fooled" VM to get hardware help in swizzling pointers while explicitly managing the buffer pool. QuickStore (see White/DeWitt in red book) goes into the details of whatthis kind of pointer swizzling requires.
- Obviously, there are tradeoffs depending on workload. Also, ObjectStore "loses control" of pages in VM, so must do page-level locking & physical logging.

Clustering:

- Objects are connected in a graph structure via pointers. Programs navigate this graph. How should you lay things out on disk to getgood locality? (see Tsangaris/Naughton SIGMOD '92 for a survey and a graph-partitioning scheme) Lots of rediscovery in file systems and web settings! Clustering is a classic problem, of course, but the disk version is treated first in OODBMSs.

Client/Server Caching & Prefetching:

- Typically OODBMSs were to be used in a client/server environment, wherethe client would operate on a portion of the database (e.g. a piece ofa VLSI diagram.) Would like to do intelligent client/server caching.
- Need to pay attention to transactions in this environment!
- Page-shipping vs. object-shipping
- Franklin et al. did good work in this area, including an excellent surveywe'll see soon.

Indexing:

- Over class hierarchies
- Over path expressions

- Some nice tradeoff papers here, nothing especially surprising.

Query Processing and Optimization:

- Declarative languages (OQL)
- Algebras that include support for complex objects (e.g. Nest/Unnest, othercomplex type constructors)
- Path expressions
- Extensible optimizer generators (Graefe's work on Exodus, Volcano, Cascades)
- O2 is the only "real" system to explore this, and the basis of much of the research (which tends to be rather fussy)

Schema Evolution, Bulk Loading:

- How to make this possible and efficient?

Benchmarking:

- Wisconsin's OO7 the de facto benchmark (pretty academic, though!)

IV. Takeaways

- Don't compete with relational (and its evolution) for the storage/query market. It's too big a task, customers are too conservative in this space. Evolvability outweighs performance for the bulk of the market. The rest of the market rolls its own solution. (XML database vendors beware!)
- A bunch of nice research was done in the OODB space. Some of it isrelevant for ORDBMS (e.g. clustering, indexing, query processing/opt). Some of it is relevant in other environments as well -- e.g. pointer swizzling is a classic problem, transactional caching may become important on the web, etc.
- The XML and "semi-structured" research has done little systems work thatdoesn't look just like OODB or relational work. (We may have a peek later in the semester, or see CS286)
- A well-educated DB researcher should know about the bag of tricks here.