AllegroCache
alpha 0.7.4
yaoodb

By
Jans Aasman,
Franz Inc.
Complexity is coming your way
Our customers have complex database problems everywhere

Sample Complex Applications

Energy
Seismic Analysis

Life Sciences
Cancer Research
Drug Discovery
Protein Folding
Pathway Modeling

Financial Services
Derivatives Analysis
Portfolio Risk Analysis
Fraud Detection
Market Modeling

Manufacturing
Product Design
Finite Element Analysis
Failure Analysis

Telco
Bandwidth Modeling
Multiplayer Gaming

Media
Digital Rendering

Computing Infrastructure

Government
Homeland Security
DOE, DOD Research: (physics, weather)

eGovernment
Big push for ORM as part of solution

- Hibernate for Java
- Versant for .NET
- Oracle’s OO extensions
- CLSQL for Lisp
- …

- However: these are user friendly thin layers on top of RDBMS that don’t solve the real complexity problem
Why a full OO,
Why Allegrocache

- If your data is best described as a complex graph
- Graph many times larger than Memory
  - > $10^8$ objects in pointer space
- Graph search & Complex queries & Inferencing & Reasoning
  - Instead of set operations
- Very heterogeneous data, possibly in multiple databases
- Object definitions often change
- Intelligent Caching
  - More reads than writes, ultra fast access to individual records.
AllegroCache from a modern database perspective

- Stand-alone & Client Server model
  - Single user on local disk
  - Multiple clients talking to server over sockets

- Commit/Rollback

- ACID
  - Atomicity (all or nothing)
  - Consistency (or rollback)
  - Isolation (multiple transactions will not interfere)
  - Durability

- Optimistic concurrency
AllegroCache for Lispers

- Persistent CLOS on all 64 and 32 bit platforms
- Lisp Btrees (previously Berkeley DB)
  - Floating cursors, multiple concurrent readers
  - Keys and Values are unsigned byte 8 arrays of unlimited size
  - Comparison functions in Lisp
  - Comprehensive marshalling package for most datatypes
  - Fine grained dynamic control over btree cache size
    - resourced blocks, almost no consing..
  - Comparable to BDB in speed and functionality
    - 130,000 key/value pairs per second for increasing keys,
      66,000 for unordered (mostly disk bound now)
Features from (lisp) programmer perspective

- MetaClass persistent-class.
- Change class-definition supported
  - lazy update of objects
- Class definitions are first class objects in AC
- Object ID's unique for the life time of the database
  - and user accessible.
- Indexed slots
- Referential integrity
  - Deleted objects are lazily and silently changed to nil in slots.
Features for programmers (cont)

- Maps (persistent hashtables) & Sets (persistent large collections of objects)
  - Transactionally safe
  - And convenient macro’s to loop over them.

- Support main Lisp datatypes
  - Strings, lists, vectors, symbols, numbers
  - Persistent objects, Maps, Sets
  - Unsigned byte 8 arrays (for your structs and non-persistent clos objects and all other data)
  - Tell us what you need and we build it for you
Features for programmers (cont)

- Several ways to retrieve objects and object ids (oid)
  - (retrieve-from-index 'person 'name "jans")
  - (doclass (e 'person)
    (when (string= (name e) "jans")
      (print)))
  - (setf person (oid-to-object 'person 100))
  - (name (country (city person)))
Features for programmers (cont)

- Prolog as efficient higher level retrieval language
- Real soon full SQL support (courtesy of Intelligent Handbook)
- Simple webbased database browser
Current todo list

- Caching strategies and user defined caching rules
- Index range queries
- Rebuilding indexes when redefining classes
- Dumping the database into a readable format
- Restore database from the dump
- Internationalization (99 % done)
- Journalaling
To Do for 1.1

- Support for automatic blobs
- User defined indexes for slots and maps
- Query language running in the cache
- Integration with other dbms
  - (automatically reading in tables from relational databases)
  - Using rdbms for secondary storage.
- A hook for marshalling your own datatypes
- Thick Client GUI for creating objects, managing users and the database.
Premature Benchmarking

- 1,000,000,000 objects in 12 hours in stand alone mode.
  - Small objects, two slots, no overflow blocks in btree.., no indexing apart from oid

- Adding objects in constant time, nearly 18,000 obj/s

- Retrieving objects constant time, independent of size..

- Lisp size doesn’t grow beyond 260 MB

- Database on disk is 97 Gb
Premature benchmarking.
AC alpha 0.7.4 vs MySQL
on 64 bit, 1.5 Ghz, 4 Gig linux machine

(defun call-data ()
  ((call_number :index :any-unique)
    action
    from_user
    to_user
    time_start
    time_spoken
    amount
    balance
    description))

CREATE TABLE call
  (call_number int
   PRIMARY KEY auto_increment,
   action int,
   from_user int,
   to_user int,
   time_start int,
   time_spoken int,
   amount int,
   balance int,
   description varchar(200))
);
Premature benchmarking.
AC alpha 0.7.4 vs MySQL
(cont)
Our expectations for raw speed

- Given our past performance on raw speed for Perl Regexp, Validating XML parser, AllegroServe, Prolog, etc
- Writing: within range of MySQL and Oracle
- Reading: Looping through all objects in AC always slower, RDBMS can often bypass btrees, read tables with fixed size..
  - RDB: good at set operations
  - OO: good at pointer operations
- Random Access, 5 to 10 times faster than RDBMS
Applications & Prototypes

- Biolingua: A frame system on top of AC, the basis of KnowOs
- Pepito: data mining package on AC*
- TellMe: personal directory for mobile phones
- Kido: Fraud detection over Call Detail Records
- CRL: P2P document server, a secure webserver.
- Boomtree: Web-based RSS reader based on Flash that can play 'podcasts' in the browser.
- Franz: Geneology Royal British Family, Tivo Box, 90,000 RSS feeds, Pandorabots, Internal CRM package, Support Database