Architecture of Popular Object-Relational Mapping Providers

Craig Russell

Mitesh Meswani

Larry White

TS-4856

Goal

How do architectures of Object-Relational Mapping (ORM) frameworks affect performance?

Agenda

Concepts of ORM

Major User-Visible Components

Major Internal Components

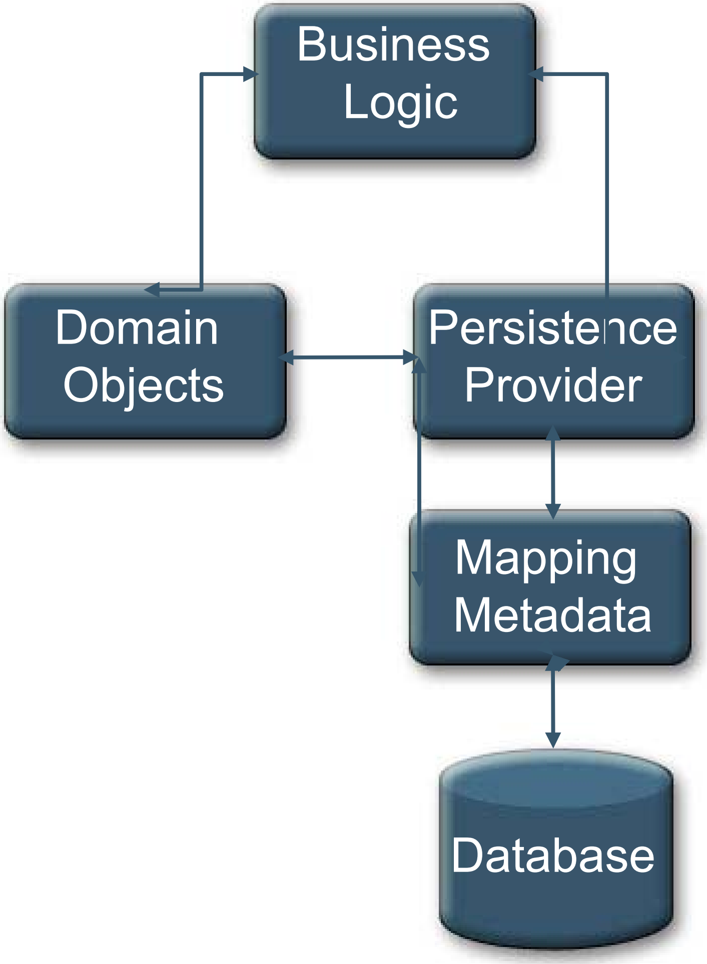
Performance Methodology

Key Findings/Recommendations

Call to Action

Concepts of ORM

- Business Logic
  - Persistence aware
  - Uses persistence API
    - Persist, remove, query
- Domain Objects
  - Plain old Java objects
  - No persistence code
- Persistence Provider
  - Implements persistence API
- Mapping Metadata
  - Domain to database



Agenda

Concepts of ORM

Major User-Visible Components

Major Internal Components

Performance Methodology

Key Findings/Recommendations

Call to Action

Persistent Class

- The domain object persisted in database
- Instances might be persistent (or not)

Persistence Unit

- Singleton pattern for VM (JNDI)
- DataSource definition
- Metadata for mapping to database
  - Entity classes
  - Rows, columns, primary/foreign keys

Persistence Context

- Represents a collection of entities managed by a persistence provider on behalf of a single user
- For each persistent identity, there is a unique entity instance in a given persistence context
- For example—Entities managed by
  - Java Persistence API (JPA) EntityManager or
  - Java Data Objects (JDO) PersistenceManager

Query API

- Provides query using domain model artifacts
  - Entity classes
  - Persistent fields/properties
  - Relationships
- Persistence provider translates to SQL
  - Or underlying native query language (QL)
  - One domain query translates to one native QL

Second Level Cache

- Managed in persistence unit
- Contains entities used at least once by any persistence context in a persistence unit
- For each persistent identity, contains the unique entity instance last updated in a persistence context
- Cached entities can become stale

Connection Pooling

- Access to database requires a connection
- Connections are expensive to acquire
- Connection is transactional (or not)
- Connections can be serially reused

Field vs. Property Persistence

- Persistent fields
  - Fields behave according to Java platform access modifiers
    - Private, protected
  - Persistence provider uses reflection or byte code Enhancement to access fields
- Persistent properties
  - JavaBeans™ architecture style get/set methods
    - Both user contract and persistence provider
    - No validation
  - Violates separation of concerns

Byte-code Enhancement

a.k.a. weaving, transforming

- Class byte codes modified for efficiency
  - Direct access to fields without reflection
  - Automatic change tracking
  - Lazy field loading
- Requires changes to deployment
  - Dynamic instrumentation via Java technology agent [Java Platform, Standard Edition (Java SE platform)]
  - Dynamic instrumentation via transformer [Java Platform, Enterprise Edition (Java EE platform)]
- Static enhancement provides more options

Agenda

Concepts of ORM

Major User-Visible Components

Major Internal Components

Performance Methodology

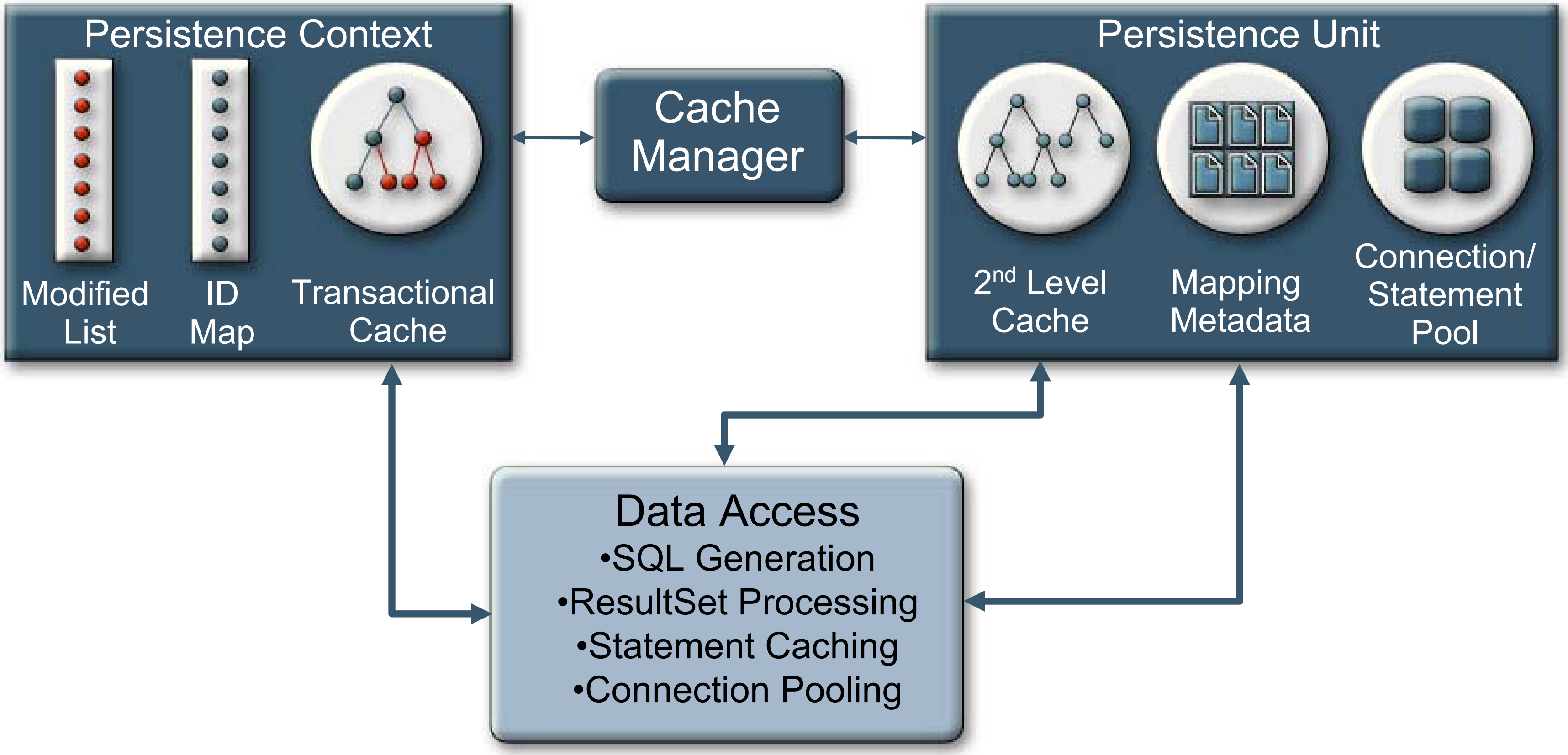
Key Findings/Recommendations

Call to Action

Major Internal Components

- Persistence unit
  - DataSource (Connection Factory)
    - Statement cache, query cache
    - Persistent class metadata
    - Second-level cache
- Persistence context
  - Entity maps/lists
    - Transactional entities: map<key, entity>
    - Modified entities: list<entity>
- SQL generator/resultSet handler

Internal Architecture



Persistence Context

Modified List

ID Map

Transactional Cache

Cache Manager

Persistence Unit

2nd Level Cache

Mapping Metadata

Connection/ Statement Pool

Data Access

- SQL Generation
- ResultSet Processing
- Statement Caching
- Connection Pooling

Lazy Loading/Change Detection

- OpenJPA uses byte code enhancement
- JPOX uses byte code enhancement
- TopLink essentials uses byte code enhancement
  - Reflection for change detection
- Hibernate uses reflection for change detection
  - Doesn't load lazily with field persistence

Agenda

Concepts of ORM

Major User-Visible Components

Major Internal Components

Performance Methodology

Key Findings/Recommendations

Call to Action

Performance Methodology

Can we really do that?

- Performance is difficult to measure (technically)
  - Choose a workload
  - Choose a configuration
  - Choose products to measure
  - Measure, verify, repeat (until exhaustion)
- Performance is difficult to measure (legally)
  - Restrictions, covenants, licenses
  - **Open Source Changes Everything**
    - Well… almost

Performance Methodology

How do we do that?

- Theorize—Predict relative performance
    - Qualitative understanding of architecture
        - Caching, data access, change detection
- Conduct experiments
- Analyze and correlate with measurements
- Modify workload and repeat

Performance Methodology

Why should we **NOT** do that?

- “All workloads are different”
  - Workloads are composites of atomic use cases
    - Typically 5 to 15 use cases
    - Find, query, retrieve object graph, update, delete
  - Vary mix and correlation
  - “eBay is different from amazon”

Performance Methodology

Why should we **NOT** do that?

- “Micro benchmarks are trivial”
  - Workloads are composites
  - Each individual operation is trivial but in aggregate represents real work
  - Quantity has a quality all its own

Performance Methodology

Why should we **NOT** do that?

- "Why should you trust a vendor?"
  - ORM frameworks are open source
    - We don't need to trust vendors
    - The community can engage this problem
    - Users can modify to suit their requirements

Measuring Performance

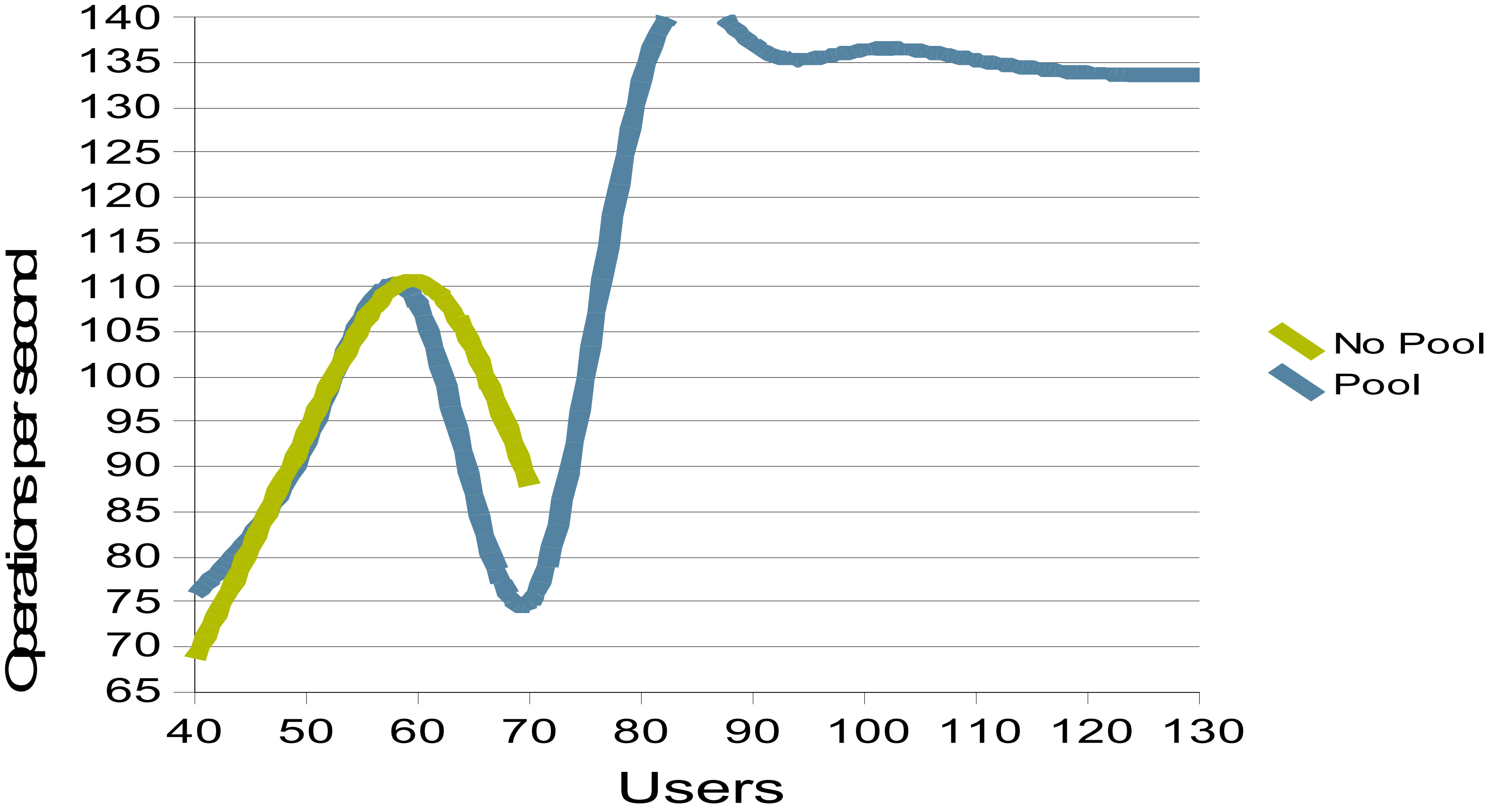
- Workload driver
  - Operations (create, read, update, query, navigate)
  - Mix (e.g. 10%, 60%, 5%, 20%, 5%)
  - Timing requirements (90% percentile within 2 seconds)
- Operations implementation
  - Use persistence adapter to do work
  - Translate operation to adapter API
- Scale number of parallel threads
  - Measure operations per second
  - Increase parallelism until metric decreases

DIY Open Source Project

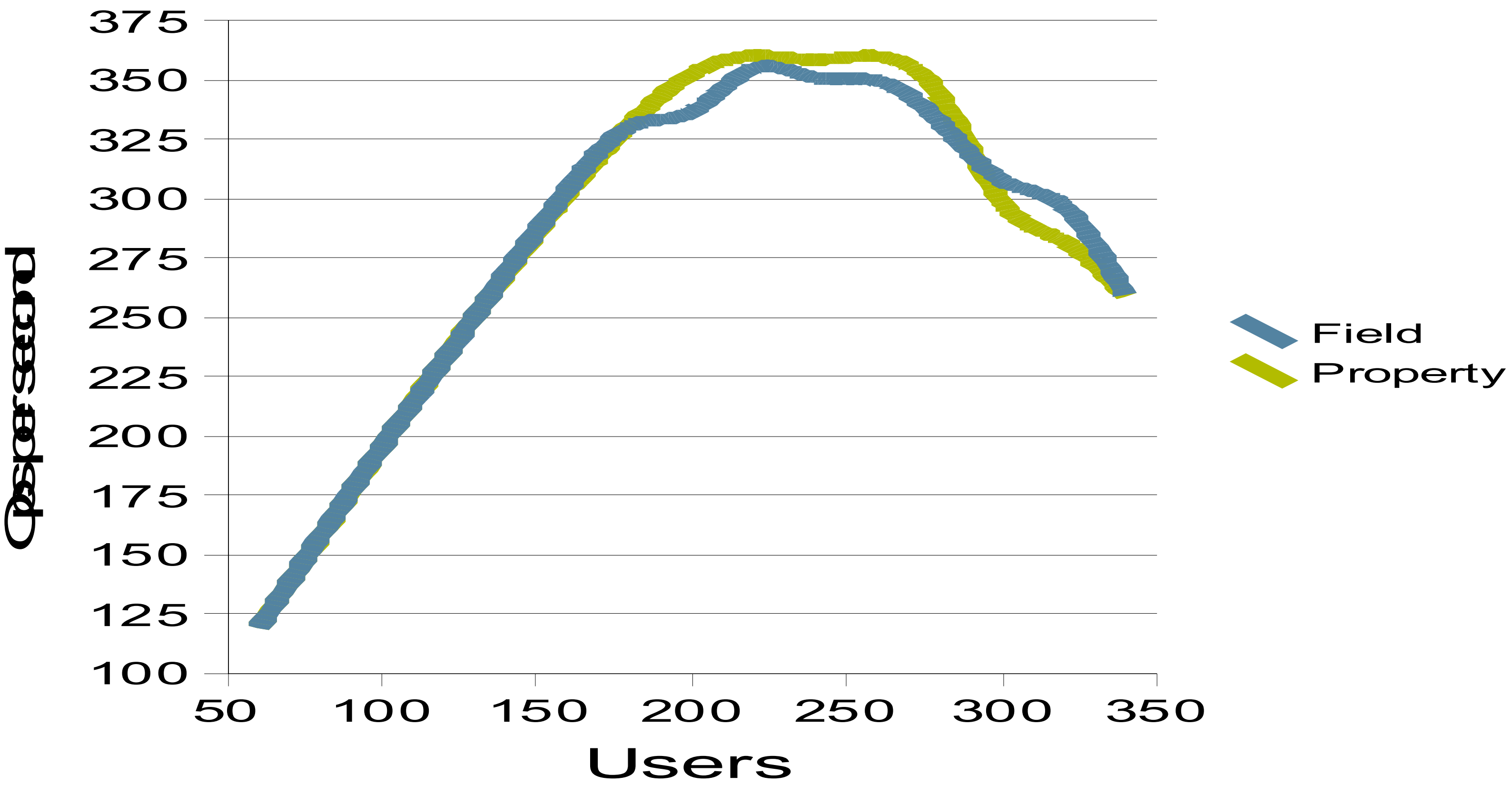
- Allows users to run their own workloads
- Sample workloads and configurations provided
- URL http://diy.dev.java.net
- Suggestions and contributions encouraged

Connection Pooling Performance

Hibernate Connection Pooling

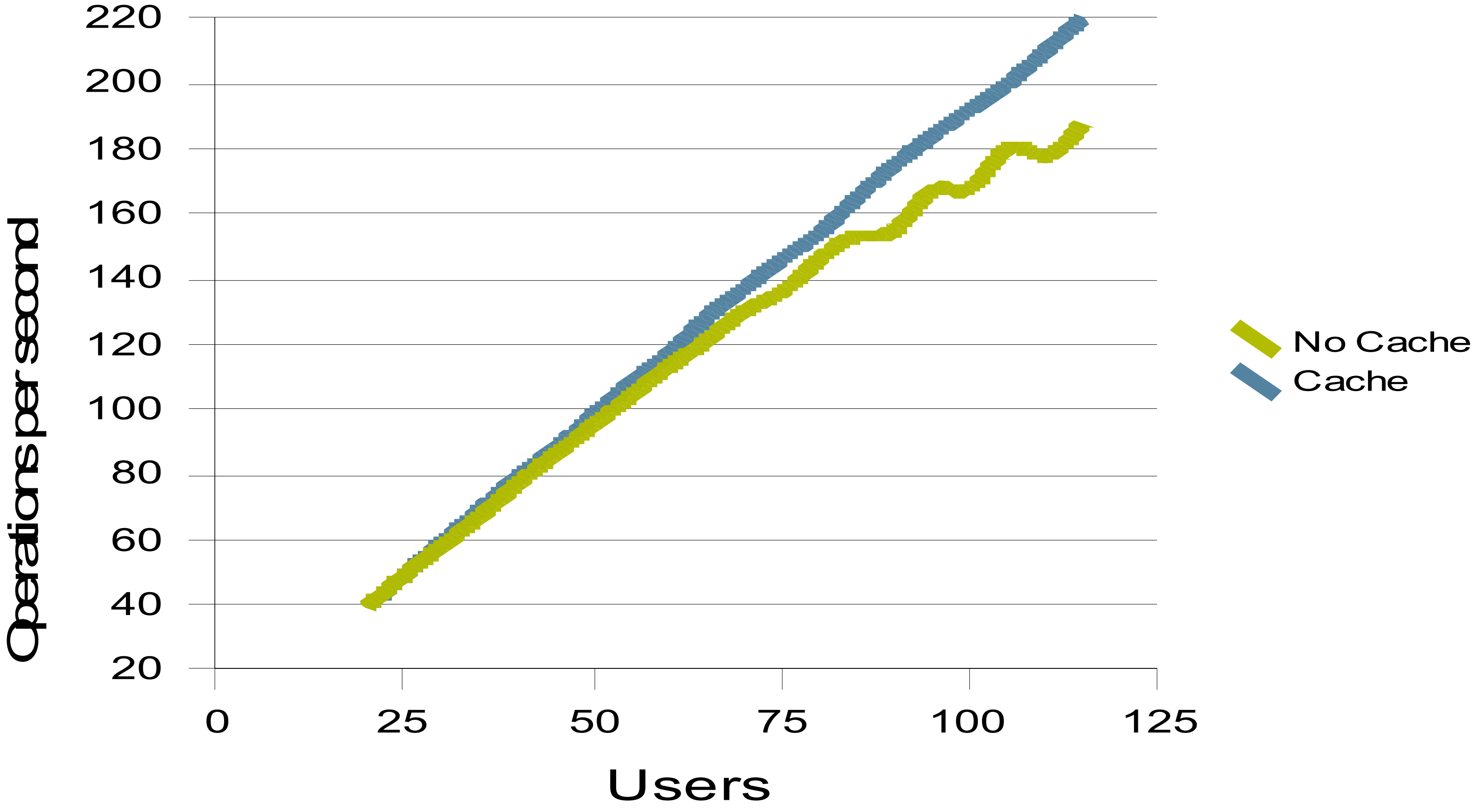


TopLink Essentials Field vs. Property



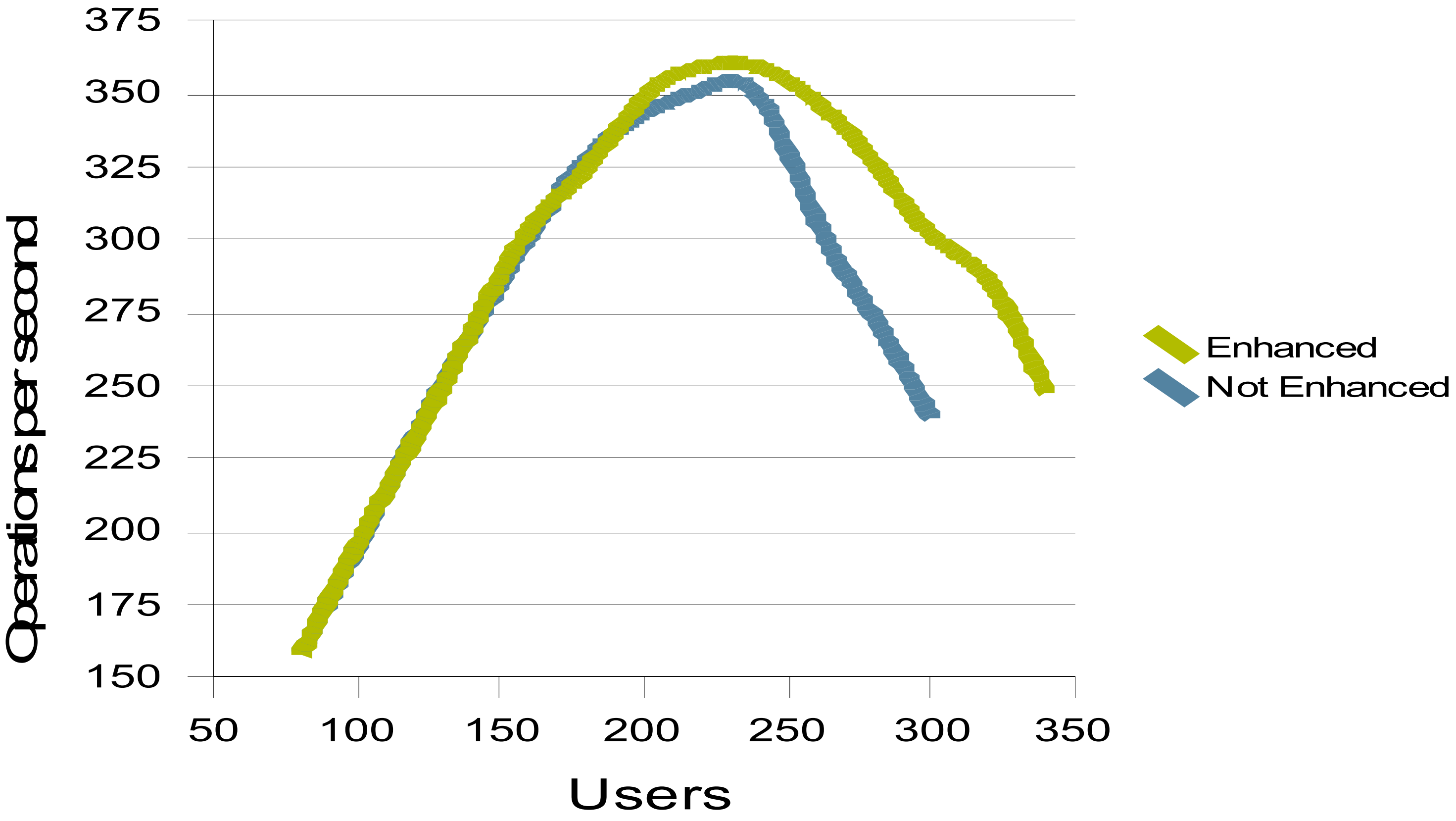
Use Second Level Cache

OpenJPA Second Level Cache



Use Byte-code Enhancement

TopLink Essentials Enhancement



Key Findings

…And recommendations

- Use second-level cache
- Use connection pooling
- Use byte-code enhancement
- Use field persistence

Call to Action

- Performance tuning is required
  - For all but the most trivial applications
- It’s much too hard to tune persistence
- Performance monitoring should be automatic
  - Should have to turn it off if you don’t want it
  - Should enable access strategy tuning
    - Part of every database access
- Take a look at DIY Project

Q&A

Craig Russell
Mitesh Meswani
Larry White

http://diy.dev.java.net

Architecture of Popular Object-Relational Mapping Providers

Craig Russell

Mitesh Meswani

Larry White

TS-4856