H2O persistence framework for column oriented distributed (NoSQL) databases

Dino Kečo, Dženana Đonko

University of Sarajevo, Faculty of Electrical Engineering

Zmaja od Bosne bb, 71000 Sarajevo, Bosnia and Herzegovina

E - mails: *dino.keco@gmail.com*, *ddonko@etf.unsa.ba*

Abstract

Cloud architectures are most commonly used in cases when large scale data processing is required. Building applications for cloud architectures requires a lot of engineering experience, especially in cases of data persistence. Persistence in cloud architectures is solved using NoSQL database models. In this paper we are working with column oriented NoSQL database model. Main research goal of this paper is building of new persistence framework for column oriented NoSQL databases. H2O (HBase to Object) framework is created to resolve problem of mapping objects into rows in column oriented database and to provide effective mechanisms for data retrieval. Main focus of this framework is to support persistence of domain models presented by standard UML language. Current implementation supports storing content into HBase NoSQL database. Core engine of H2O framework is built on top of XPath standard. All mappings between domain model attributes and columns in row are represented using XPaths. These paths are used to transform object into row and vice versa. H2O framework contains component for integration with Hadoop map reduce processing library to simplify writing of Hadoop map reduce parallel programs. We took two hardware platforms of same price. First platform have HBase 0.90.1 and H2O installed and other have installed Oracle 11g and Hibernate framework. We are comparing performance of these two platforms from aspects of retrieval and persistence of objects. Result of our comparison is that NoSQL model is better from aspects of retrieval by primary key but shows lower performances in save operations.

Keywords: NoSQL, persistence, distributed, HBase, Hadoop, mapping, framework, UML, map-reduce

1.INTRODUCTION

Problem of mapping and persistence of objects in relational database model was open question for about 15 years [1]. This problem is resolved by ORM frameworks like Hibernate. Mapping and persistence of objects into NoSQL database model is even harder to solve because difference between models is much larger. Our work is focused on developing framework which will resolve these two problems.

H2O framework is object/row mapping tool that provides user friendly interface to persistence application layer. This interface is developed using DAO (data access object) design pattern.

Because NoSQL database model is easy to integrate with map reduce programs, in this paper we present H2O modules which are used for integration with Hadoop map reduce library [5]. Mainly these modules are used to simplify process of creating map reduce jobs.

This paper makes the following research contributions:

We present model of new persistence framework.

We present implementation of H2O framework with support for HBase database.

We present modules for integration with Hadoop map reduce library.

Section 2 provide more detailed explanation of mapping and persistence problem. In section 3 we present model of H2O framework. Section 4 provides implementation details with focus on main components. We present model and implementation of components used for integration with map reduce library in section 5 and we conclude in section 6.

2. PROBLEM FORMULATION

All business applications have domain model which is presented by graph of classes. In most cases these classes are presented using UML modeling language. Main persistence problem is mapping of graph of objects into format suitable for storing. In this case we are mapping graph of objects into key value database storage.

ORM problems like granularity, subtypes, identity, data navigation, relation to association, etc. [1] are even harder to solve because data models are much more different. Concepts like inheritance, encapsulation and polymorphism doesn't exist in key value storages and because of that it is necessary to find appropriate replacement for those concepts. What H2O is trying to resolve is illustrated on Fig 1. and model of one solution, based on xpath standard [6], is presented in next section of this paper.

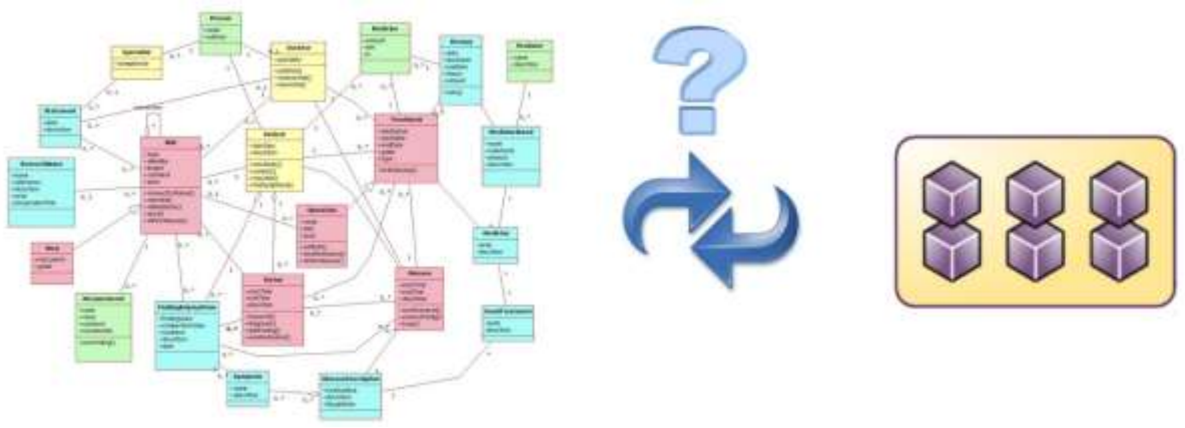

Fig 1. Conceptual illustration of mapping problem for key value distributed storages

On the left side of Fig 1. we have graph of domain objects, while on the right side we have key value database storage. H2O needs to find best possible way to map data presented like graph of objects into row of key values storage and to keep data consistent in any possible case.

3. FRAMEWORK MODEL

In this section we present model of H2O framework and it basic components which are presented on Fig 2.

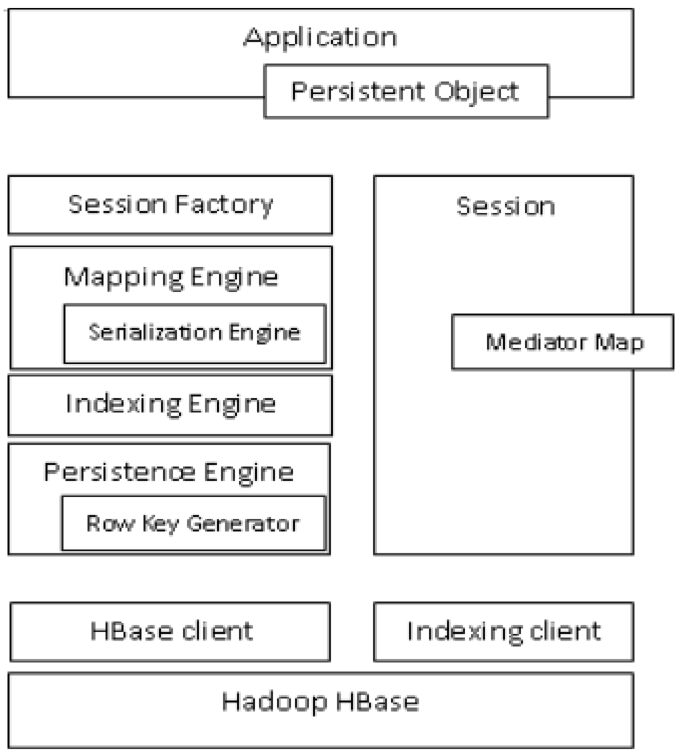

Fig 2. H2O basic components

Entry point component for user of H2O is Session, which provides basic methods (get, create, update and delete) to work with database. Main component of H2O framework is MappingEngine which performs conversion of graph of objects into key value row and vice versa. IndexingEngine is component which provides support for indexing. PersistenceEngine is component which is used for interaction with specific database implementation. In this case we are using HBase persistence engine. IndexingEngine and PersistenceEngine should be implemented by user for specific database and index implementations. MappingEngine, most important and most complicated, component is based on customized xpath standard which is capable to describe additional information about each path in graph of objects. This customization is needed to describe data types (class) of nodes in graph of objects.

Problem of subtypes, which is main problem in ORM, is resolved in H2O by creating different mappings for each of subtypes available. Although ORM persistence frameworks have multiple strategies for persistence of subtypes [1] all of them are compatible with each other. Because of that fact in H2O we are supporting just one mapping strategy for subtypes.

If there is a need for cycles in domain model that can't be resolved using xpath because xpath is structure driven, which causes infinite loops in mappings. Because of this new component is introduced, SerializationEngine, which uses one of standard serialization frameworks for data serialization. Introduction of serialization enables H2O to handle cycles because serialization frameworks are data driven [7]. Any user of H2O framework can easily implement his own strategy of data serialization by extending interface of SerializationEngine.

4. IMPLEMENTATION

In this section, we are presenting API which provide H2O framework for data persistence and data retrieval. As part of this section we present basic comparison of persistence performances between H2O/HBase and Hibernate/Oracle databases. For each comparison set we have used same hardware platforms. We have used PowerEdge M805 Dell Blade servers with two Quad Core Xeon processors and 128 GB of RAM memory. For Hibernate/Oracle test we have used two M805 servers connected to Oracle cluster, and for H2O/HBase we have used VM Ware virtualization and created 8 hosts for setting up HBase cluster. Each node had 2 cores and 32 GB of RAM memory. Oracle RAC version 11g is used for setting up Oracle database while HBase 0.90.1 version is used for HBase cluster.

Even if there is a big difference between relational and key/value database models, API on DAO layer is same for any database if DAO design pattern is used in application architecture. H2O framework is built for applications which will use DAO design pattern in their architecture. Main advantage of this is that all business application can be modeled using standard modeling languages like UML and persist that model into any type of database.

On Fig 3. we present DAO API which is provided by H2O framework for any kind of database implementation.

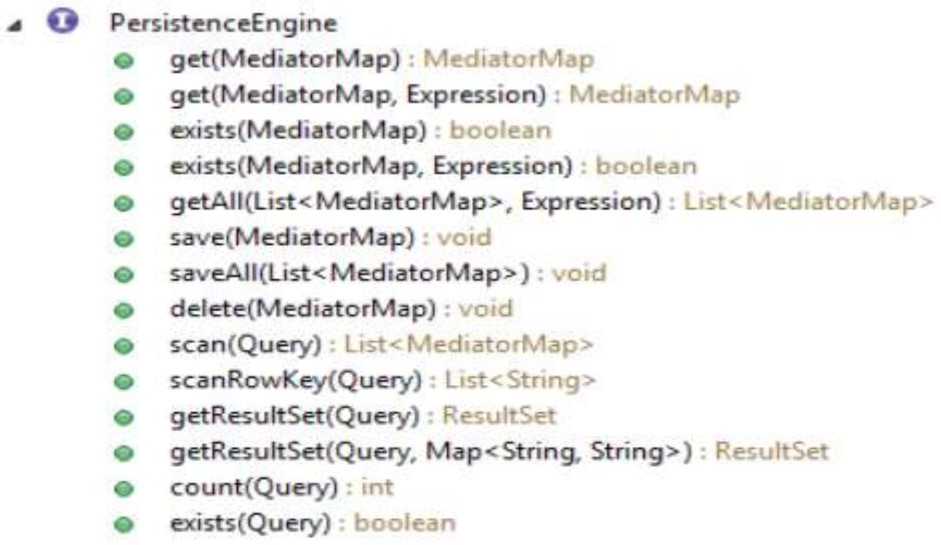
```
PersistenceEngine
    get(MediatorMap) : MediatorMap
    get(MediatorMap, Expression) : MediatorMap
    exists(MediatorMap) : boolean
    exists(MediatorMap, Expression) : boolean
    getAll(List<MediatorMap>, Expression) : List<MediatorMap>
    save(MediatorMap) : void
    saveAll(List<MediatorMap>) : void
    delete(MediatorMap) : void
    scan(Query) : List<MediatorMap>
    scanRowKey(Query) : List<String>
    getResultSet(Query) : ResultSet
    getResultSet(Query, Map<String, String>) : ResultSet
    count(Query) : int
    exists(Query) : boolean
```

Fig 3. API provided by H2O framework

As shown in Fig 3. H2O provides basic CRUD (Create, Read, Update, Delete) operations.

We have performed three tests to compare Oracle database and Hibernate as persistence framework and HBase and H2O on the other side. We have performed following tests:

Data retrieval by primary key: In this experiment we monitor speed to read one record (record size 2KB) from database by primary key. We performed these tests with different number of records persisted in databases. Results of this tests are presented on Fig 4. As shown on Fig 4. Relational database doesn't scale well when # of records is greater than 64 M.

Data persistence with increasing number of objects in graph: In this experiment we monitor speed to persist one record but with variable record size (nodes in graph). Results on Fig 5. shows that HBase/H2O is much slower than relational database, which is caused by MappingEngine. MappingEngine component doesn't have multi-threaded processing and that is main reason for slowness. Definitely there is open space for optimization of this component.

Data persistence with static number of objects in graph: In this test we are inserting records in database with variable number of records inside database. Results on Fig 6. shows that insert in database is static regarding number of records persisted in database.

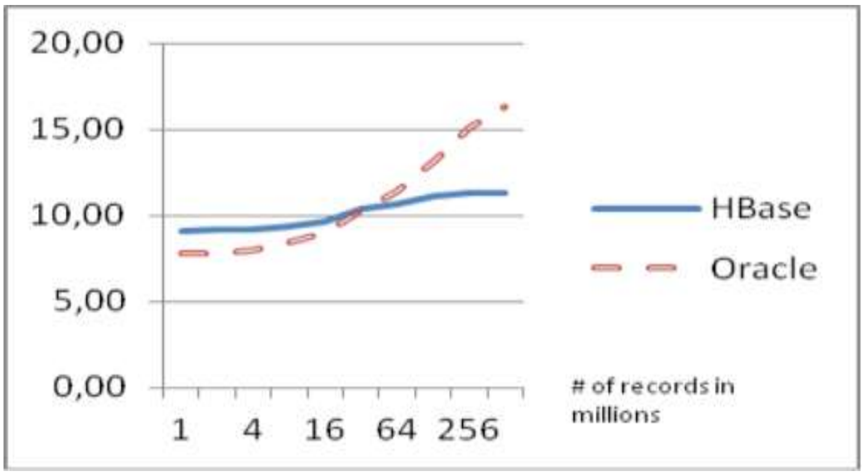

Fig 4. Data retrieval by primary key

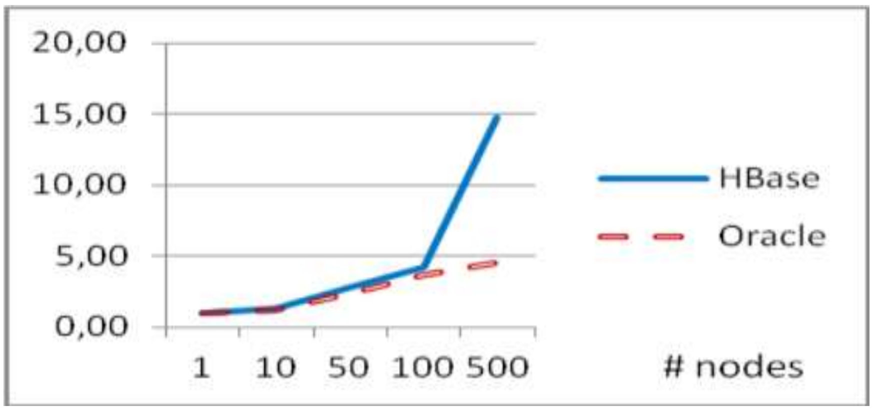

Fig 5. Data persistence with increasing number of nodes in domain graph

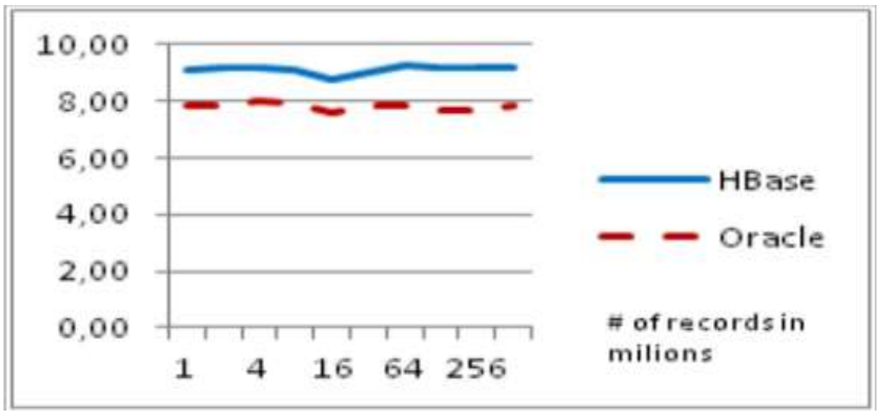

Fig 6. Data persistence with static number of nodes in domain graph

5. MAP REDUCE INTEGRATION

Inspired by the map and reduce primitives present in functional languages, Google proposed the Map Reduce [3] abstraction that enables users to easily develop large-scale distributed applications. Mechanisms of fault tolerance is handled inside of map reduce library by re-executing failed tasks.

In this model, the computation inputs a set of key/value pairs and produces a set of output key/value pairs. The user of the map reduce library expresses computation as two functions: Map and Reduce. Map written by user, takes an input pair and produces a set of intermediate key/value pairs. The map reduce framework then groups together all intermediate values associated with same intermediate values key and passes them to reduce function. The Reduce function, also written by user, accepts intermediate key I and set of values for that key. It merges together these values to form a possibly smaller set of values.

H2O framework goes one step more in abstraction over map reduce framework. Custom adapters inside H2O framework enables that value inside key/value pair be a graph of objects. This provides user more flexible and more user friendly interface to work with. This enables faster development of map reduce applications where complexity of persistence and mappings is hidden inside of H2O framework.

Because, H2O framework is build to work with Hadoop implementation of map reduce library, adapters for integration, known as H2OInputFormat and H2OOutputFormat are named by Hadoop naming standard.

6. CONCLUSION AND FUTURE WORK

H2O is created to solve problem of mapping between graph of domain objects and row in key/value storage. As presented in this paper there is a lot of open space for improvements in MappingEngine component. Also support for other implementations of key/value storages like Cassandra should be implemented.

Main reason why H2O is created is to speed up development process by using standard UML modeling techniques and to solve all problems related to persistence. This will enable users to focus on business logic instead of technical details. Integration with Hadoop map reduce library provides easy way to write parallel applications and not even to worry about data persistence.

For future we plan to create an open source project from H2O to involve more people into this and to gather new ideas.

REFERENCES

Christian Bauer and Gavin King (2006.) - Java Persistence with Hibernate Second Edition of Hibernate in Action, Manning,

Ming-Yee Iu and Willy Zwaenepeol - HadoopToSQL a MapReduce Query Optimizer, EuroSys (2010),

Jeffrey Dean and Sanjay Ghemawat - MapReduce: Simplifed Data Processing on Large Clusters, OSDI (2004), HBase web page - hbase.apache.org,

Hadoop web page - hadoop.apache.org,

Jxpath web page - commons.apache.org/jxpath,

JacksonJSON web page - jackson.codehaus.org