УДК 004.652.6 **Технические науки**

Селиванов Павел Александрович, *студент-магистр, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

Белов Юрий Сергеевич, *к.ф. -м.н., доцент, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

ХРАНЕНИЕ ОБЪЕКТОВ В FOUNDATIONDB НА ОСНОВЕ МАТЕРИАЛИЗОВАННОГО ПУТИ

Аннотация: Данная статья дает представление о нереляционной базе данных FoundationDB. Рассматривает вариант хранения данных на основе материализованного пути. Говорится о хранении строковых, числовых, логических переменных, а также о сериализации обычных и ассоциативных массивов.

Ключевые слова: FoundationDB, нереляционная база данных, хранение объектов, сериализация, десериализация.

Annotation: This article provides an introduction to the FoundationDB non-relational database. Considers a variant for storing data based on a materialized path. We are talking about storing string, numeric, boolean variables, as well as serializing regular and associative arrays.

Key words: FoundationDB, non-relational database, object storage, serialization, deserialization.

FoundationDB — это распределенная база данных, предназначенная для обработки больших объемов, структурированных данных в кластерах серверов. Она организует данные в качестве упорядоченных пар ключ-значение и

использует транзакции ACID для всех операций. Эта база данных особенно хорошо подходит для нагрузок чтения / записи, но также имеет отличную производительность при интенсивных операциях записи [1].

Стандартный интерфейс базы данных не предусматривает работы с объектами, он умеет оперировать только с массивами байт. Как имея только массивы байт организовать хранение различных объектов [2]? Для решения этой задачи можно в ключе расположить материализованный путь к объекту.

Если посмотреть на то, как организовано хранение в SQL, можно увидеть базу, таблицу, поля – это и есть ни что иное, как путь до нужной информации.

Для того чтобы на основе одного кластера FoundationDB можно было организовать несколько независимых баз данных, добавим в ключ каждого объекта признак базы, это может быть, как строка, преобразованная в байты, так и цифра. Можно взять 1 байт и это позволит нам разместить 255 различных баз или 2 байта – 65636. Конечно, для идентификации базы нужно выделить какое-то разумное кол-во бит, ведь это значение будет дублироваться для каждого поля объекта.

Каждая база данных создается для хранения объектов одного типа в отдельных таблицах. Признак таблицы также записывается в путь до объекта. Тут работают аналогичные соображения что и для базы данных – нужно стремиться к сокращению объема этого признака [3].

После связки базы данных и таблицы идет идентификатор объекта. Это значение должно быть уникальным в рамках таблицы. Например, для этого можно использовать целое положительное число и каждый раз увеличивать его на единицу или UUID. У первого варианта есть свои недостатки: при каждом сохранении необходимо делать выборку объектов и смотреть идентификатор самого последнего объекта (все ключи в FoundationDB хранятся лексикографическом порядке, то есть отсортированы по умолчанию в порядке возрастания), а при параллельной записи возможна ситуация, когда две независимые транзакции сделают выборку, найдут одинаковый идентификатор и при фиксации одна из них завершится неудачно из-за конфликта. Второй

вариант решает эту проблему, но приходится платить объемом – размер ключа возрастает.

На данном этапе получается, что если база идентифицируется как db, таблица – table, идентификатор объекта – 1, то путь (ключ объекта) к объекту представляет собой следующую цепочку (элементы разделены / для облегчения понимания):

```
db/table/1
```

В самом простом варианте можно сериализовать объект, например, используя json, и сохранить как значение, но у этой реализации есть свои минусы: на сериализацию/десериализацию тратятся ресурсы, нет возможности частичного обновления объекта, размер ключа в FoundationDB ограничивается 100 килобайтами (При этом для оптимальной производительности разработчики рекомендуют значения длиной 10 килобайт) [4].

При такой реализации объект сохраняется следующим образом:

```
db/table/1 : {field1:”fdsfs”, field2:123};
```

Продолжив развитие концепции материализованного пути, можно решить часть описанных выше проблем. Идея заключается в том, чтобы хранить объект не в одной паре ключ/значение, а в нескольких. Для решения проблемы ограничения размера значения можно сериализованный объект разбивать на части и сохранять в разные пары ключ/значение, но для реализации такого механизма потребуется очень много усилий, среди которых дополнительные параметры в каждом ключе и достаточно сложная логика сохранения. Более простой вариант решения этой проблемы является отказ от сериализации объекта целиком в пользу сериализации по полям объекта.

Каждое поле объекта сохраняется отдельной парой ключ/значение. Допустим объект «Пользователь» имеет 2 поля: имя и возраст, тогда объект сохраняется следующим образом:

```
db/table/1/name : Ivan;
db/table/1/age: 26;
```

Название поля, которое помещается в путь должно быть уникальным в пределах таблицы и иметь минимальный размер для более эффективного хранения.

Необходимо понимать, что данная концепция хранения не очень подходит для вложенных структур и при проектировании объекта нужно это учитывать.

Для того чтобы выбрать объект из базы потребуется запросить подобласть ключей:

```
db/table/1/*
```

FoundationDB проявляет своё быстродействие как раз в таких ситуациях, если использовать ее для выборки единичных записей ключей/значение, то не получится раскрыть весь потенциал этой мощной базы данных. Она предоставляет возможность работы с feature объектом, суть которого заключается в том, что он возвращает нужные вам записи сразу, но реальная их подгрузка происходит позже незаметно для клиента. После получения всех записей ключ/значение требуется лишь собрать объект по кусочкам.

На «ручную» сериализацию полей тратится не так много ресурсов. Для работы достаточно разработать набор методов упаковки используемых типов данных. Реализация этих методов зависит от языка программирования, но стоит рассмотреть упаковку обычного и ассоциативного массива.

При работе с массивом необходимо сохранить порядок элементов. Массив размера N сохраняется как N пар ключ/значение. Массив [red, green, blue] сохраняется следующим образом:

```
db/table/1/colors/0 : red;
db/table/1/colors/1 : green;
db/table/1/colors/2 : blue;
```

При работе с ассоциативным массивом необходимо не забывать про уникальность ключа. Аналогично обычному массиву создается N пар ключ/значение Массив {“key1”:”value1”, “key2”:”value2”} сохраняется следующим образом:

```
db/table/1/map/key1 : value1;
db/table/1/map/key2 : value2;
```

Для реализации корректной десериализации необходимо ввести разделители элементов материализованного пути, либо зафиксировать размер его элементов, чтобы была возможность однозначно определять границы поля объекта и данных если, они будут попадать в ключ.

Таким образом, аккумулируя все вышеописанное, объект базы данных db1, таблицы players:

```
{
"nickname": "Alex",
"level": 26,
"active": true,
"powers": ["Immortality", "Teleportation"]
}
```

будет сохранен, как набор следующих записей:

```
db1/players/nickname : alex;
db1/players/level : 26;
db1/players/active : t;
db1/players/powers/0 : Immortality;
db1/players/powers/1 : Teleportation;
```

Предложенная модель хранения объектов дает возможность построения нескольких баз данных на основе одного кластера, разграничивает хранение объектов в рамках конкретной базы данных, частично решает проблему ограничения размера значения, производит достаточно быструю и малозатратную сериализацию/десериализацию как по ресурсам, так и по сложности, использует преимущества FoundationDB в плане выборки подобласти ключей, позволяет производить как полное, так и частичное обновление объекта.

Библиографический список:

1. Селиванов П.А., Белов Ю.С. ОБЗОР FOUNDATIONDB. В сборнике: Сборник избранных статей по материалам научных конференций ГНИИ "Нацразвитие". Материалы Международных научных конференций. 2020. С. 107-109.
2. И. Ю. Парамонов, В. А. Смагин, Н. Е. Косых, А. Д. Хомоненко. Методы и модели исследования сложных систем и обработки больших данных: монография /; под редакцией В. А. Смагинаи А. Д. Хомоненко. — Санкт-Петербург: Лань, 2020. — 236 с. — ISBN 978-5-8114-4006-1. // URL: https://e.lanbook.com/book/126938.
3. Тарасов, С. В. СУБД для программиста. Базы данных изнутри / С. В. Тарасов. — Москва: СОЛОН-Пресс, 2015. — 320 с. — ISBN 978-2-7466-7383-0. // URL: https://e.lanbook.com/book/64959.
4. Эрик, Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / Р. Эрик, Р. У. Джим; под редакцией Ж.

Картер ; перевод с английского А. А. Слинкин. — Москва: ДМК Пресс, 2013. — 384 с. — ISBN 978-5-94074-866-3. // URL: https://e.lanbook.com/book/58690.