УДК 004.657 DOI 10.21778/2413-9599-2017-4-125-134

Ю. А. Григорьев[1], А. Д. Плутенко[2], А. В. Бурдаков[3], Е. В. Цвященко[4]

[1] Московский государственный технический университет им. Н. Э. Баумана, Москва, Россия, [2] Амурский государственный университет, Благовещенск, Россия, [3] Информационные технологии для эпидемиологии, Москва, Россия, [4] Научно-производственная компания «Кронос-Информ», Москва, Россия

АНАЛИЗ ПРОЦЕССОВ СОГЛАСОВАНИЯ ВЕРСИЙ ЗАПИСЕЙ В БАЗАХ ДАННЫХ NOSQL

В настоящее время наряду с реляционными базами данных стали широко использоваться базы данных NoSQL. Они обладают высокой производительностью и надежностью, так как здесь данные хранятся в виде записей <ключ, значение>, которые многократно реплицируются. Но отсутствие в этих базах явного механизма ведения транзакций и блокировок записей приводит к резкому увеличению числа версий записей в базе данных при одновременном их обновлении несколькими пользователями. При этом растет нагрузка на пользователя и время согласования этих версий, уменьшается производительность системы, увеличивается вероятность потери версий. Количество типов записей, хранящихся в базе данных, велико и число версий записей невозможно измерить на этапе эксплуатации системы. Поэтому актуальной является задача оценки числа версий записи и времени их согласования при пиковой нагрузке на систему на этапе ее проектирования. Это позволяет вовремя принять правильное решение. В этом случае адекватные математические модели являются незаменимым инструментом анализа. В статье рассмотрен процесс обработки (согласования) версий записи. Разработана имитационная модель согласования версий записи, параллельно обновляемой несколькими пользователями. Модель позволяет оценить время согласования версий записи и число версий, одновременно хранящихся в базе данных. По результатам моделирования проектировщик системы может дать рекомендации о максимально возможном числе пользователей (или приложений), одновременно работающих с одним документом (записью базы данных). Это важно, если накладываются ограничения на время согласования документа. Описывается процесс подготовки и проведения натурных экспериментов в облаке для анализа адекватности модели. Результаты натурных экспериментов показывают, что разработанная имитационная модель процесса ведения версий записи является адекватной. Средняя относительная погрешность оценки характеристик случайного числа версий записи с помощью модели составляет: среднего значения – 7,5% и правой границы доверительного интервала (ПГДИ) – 3,0%. Средняя относительная погрешность оценки с помощью модели среднего значения времени обработки обновлений записи составляет 0,22%, ПГДИ – 5,6%.

Ключевые слова: *NoSQL, версии записи, вектор часов, модель согласования, адекватность.*

Для цитирования: Григорьев Ю. А., Плутенко А. Д., Бурдаков А. В., Цвященко Е. В. Анализ процессов согласования версий записей в базах данных NoSQL // Радиопромышленность. 2017. № 4. С. 125–134.

Yu. A. Grigorev[1], A. D. Plutenko[2], A. V. Burdakov[3], E. V. Tsvyashchenko[4]

[1] Bauman Moscow State Technical University, Moscow, Russia, [2] Amur State University, Blagoveshchensk, Russia, [3] Information Technologies for Epidemiology, Moscow, Russia, [4] Research and Production Enterprise Kronos-Inform, Moscow, Russia

ANALYSIS OF RECORD VERSIONS RECONCILIATION PROCESSES IN NOSQL DATABASES

At present, along with relational databases, the NoSQL databases have become widely used. They have high performance and reliability, because the data are stored in form of records <key, value>, which are repeatedly replicated. However, absence of an explicit mechanism for conducting transactions and locks of records in these databases leads to a sharp increase in the number of versions of records in the database, while they are updated by several users. In this case, the user workload and time of reconciliation of these versions are growing up, the system performance decreases, and the

probability of version loss increases. The number of types of records stored in the database is large, and the number of versions of records cannot be measured at the stage of system operation. Therefore, the actual task is to estimate the number of versions of the record and the time of their reconciliation at the peak load on the system at the stage of its design. This gives a possibility to make a right decision in time. In this case, adequate mathematical models are an indispensable tool for analysis. The article discusses the process of processing (reconciliation) of the record versions. A simulation model has been developed for reconciling the versions of the record, which is updated in parallel by several users. The model allows estimating the time of agreement between the versions of the record and the number of versions simultaneously stored in the database. Based on the results of simulation, the system designer can give recommendations on the maximum possible number of users (or applications) simultaneously working with one document (a database record). This is important if restrictions are placed on the time of document reconciliation. The process of preparation and carrying out of full-scale experiments in the cloud for analysis of model adequacy is described. The results of full-scale experiments show that the developed simulation model of the process of keeping the versions of the record is adequate. The average relative uncertainty in estimating the characteristics of the random number of versions of the record by the model is: for the mean value – 7,5% and the right confidence limit (RCI) – 3,0%. The average relative uncertainty of the estimation by the mean value of the update processing time is 0.22%, RCI – 5,6%.

Keywords: *NoSQL, record versions, model of reconciliation, adequacy.*

For citation: Grigorev Yu. A., Plutenko A. D., Burdakov A. V., Tsvyashchenko E. V. Analysis of record versions reconciliation processes in NoSQL databases. Radiopromyshlennost, 2017, no. 4, pp. 125–134 (In Russian).

DOI 10.21778/2413-9599-2017-4-125-134

Введение

Для повышения производительности и отказоустойчивости информационных систем в настоящее время все чаще используются системы баз данных, построенные на парадигме распределенных хранилищ «ключ/значение». Они получили название *NoSQL* (*Not-Only-SQL*) [1]. Эти системы являются распределенными и позволяют обрабатывать очень большие объемы данных (*Big Data*), которые накоплены, в частности, в радиопромышленности.

Только небольшое число баз данных *NoSQL* поддерживает *ACID*-транзакции в полной мере. Примерами являются графовые базы данных *Sesame* и *Neo4j* [2, 3].

Большинство баз данных *NoSQL* поддерживает *ACID*-транзакции не в полной мере. При их использовании возникают проблемы согласования реплик записей и ведения транзакций [2–4]. Примерами таких баз данных являются *Riak*, *Amazon Dynamo*, *Cassandra*, *MongoDB* и др. [2]. В них свойство атомарности (*A*) можно обеспечить только на уровне агрегата, сохраняемого в одной записи. И оно нарушается, если атомарность необходимо обеспечить на уровне нескольких агрегатов, сохраняемых в разных записях БД. В этих *NoSQL* отсутствуют блокировки. Это не позволяет поддерживать свойство изолированности (*I*) процессов, обновляющих одни и те же записи.

При отсутствии механизма блокировок записей БД пользователи *NoSQL* могут читать, а потом одновременно обновлять запись с одним и тем же ключом. В этом случае система будет хранить несколько версий данной записи. Возникает проблема согласования версий (конфликт обновления). Мотивацией выполненных в настоящей статье исследований стало увеличение числа приложений, в которых задача исследования конфликта обновления стала актуальной [2]. Ниже приведены примеры таких приложений:

- Большой коллектив сотрудников одновременно работает над проектным документом.
- В период эпидемии специалисты со всего мира срочно обсуждают диагноз заболевания.
- При наплыве покупателей часто обновляется остаток товара на складе.
- Большое число пользователей обсуждают какое-либо актуальное событие в блоге.

При чтении пользователь получает все версии записи с данным ключом, обрабатывает их и сохраняет новую версию записи. При этом старые версии удаляются из хранилища. Проблема заключается в том, что при увеличении числа версий записи растет количество анализируемых вариантов. При этом нагрузка на клиента и время согласования версий могут стать недопустимыми. Более того, возникают проблемы, связанные с ведением большого числа версий в базе данных *NoSQL* (уменьшается производительность, увеличивается вероятность потери версий) [5, 6]. Так как число записей в *NoSQL* велико, то диагностировать и устранять эти проблемы на работающей системе непросто. Целесообразно оценивать число версий записей при пиковой нагрузке на этапе проектирования. Это позволяет вовремя принять правильное решение. В этом случае адекватные математические модели являются незаменимым инструментом анализа.

В аналитической модели практически невозможно описать корреляционные зависимости между количеством клиентов, числом версий

записи и временем их обработки. Поэтому был выбран имитационный подход.

В статье рассматривается разработанная имитационная модель на языке *GPSS* [7] процесса ведения версий записи и анализируется ее адекватность по результатам натурных экспериментов. Эта модель имеет следующую особенность: она моделирует поведение вектора часов без имитации хранения самих записей. Это позволяет существенно упростить модель, при этом точность моделирования не уменьшается.

С помощью модели можно исследовать время согласования версий записи и число версий, одновременно хранящихся в базе данных *NoSQL*. Это позволяет проверить, являются ли число версий и время согласования данных допустимыми. По результатам моделирования можно дать рекомендации о максимально возможном числе пользователей (или приложений), одновременно работающих с одним документом.

С помощью разработанной адекватной модели можно оценивать указанные показатели на этапе проектирования системы *NoSQL*. Это позволяет избежать ручного подбора значений требуемых параметров для большого числа типов записей БД на этапе наладки системы. Также отпадает необходимость натурного моделирования экстремальной нагрузки на систему.

Анализ публикаций по теме исследования

В [2, 8] рассматриваются пессимистический и оптимистический подходы к обеспечению согласованности обновлений при параллельной работе. Пессимистический подход ориентирован на предотвращение возникновения конфликтов обновления. Оптимистический подход допускает возникновение конфликтов, но позволяет их обнаружить и предлагает способы их устранения.

Во многих базах данных *NoSQL*, явно не поддерживающих блокировки обновляемых записей, пессимистический подход реализован путем реализации репликации записей по схеме «*master–slave*» (рис. 1).

Здесь запрос на обновление записи (*put*) переадресуется узлу, где хранится *master*-реплика. Этот метод обеспечивает последовательное согласование, и все обновления будут появляться на узлах в одном и том же порядке. Но репликация «*master–slave*» имеет существенные недостатки [3, 8, 9]: снижается производительность, может быть нарушен доступ к *master*-реплике (часто при сбое первичного узла вторичные узлы не могут «договориться» о новом первичном сервере при нестабильной задержке в сети). Схема репликации «*master–slave*» реализована в *NoSQL MongoDB* [10], *Amazon DynamoDB* [11], *Redis* [12] и др.

Оптимистический подход к обеспечению согласованности обновлений при параллельной работе рассмотрен в работах [2, 3, 8, 13]. В [8] отмечаются преимущества оптимистического подхода: он позволяет улучшить доступность, т.к. здесь нет одной точки обновления, обеспечивает небольшое время синхронизации, и поэтому позволяет добавлять большое число копий.

Наиболее интересное применение оптимистического подхода в *NoSQL* связано с использованием версий записей в схеме обновления «*master–master*» (одноранговая репликация) (рис. 2).

Здесь все узлы, которые хранят реплики записи, являются первичными (*master*). Они могут одновременно принимать запросы на обновление одной и той же записи (*put*). Так как запросы поступают одновременно и узлы не могут распознать причинно-следственные связи этих обновлений, то одна и та же запись сохраняется и потом реплицируется в виде версий (версия 1 и версия 2). Каждая версия снабжается вектором часов [1, 6, 14, 15]. Запрос на чтение (*get*) получает все версии. Перед сохранением новой версии записи (версия 3) автоматически формируется ее вектор часов. Для этого используются векторы часов исходных прочитанных версий записи (т.е. версии 1 и версии 2). При поступлении записи на узел (узел 2) система анализирует векторы часов всех версий этой записи. Она выявляет причинно-следственную связь между версией 3 и версиями 1, 2. Далее старые версии 1, 2 удаляются на узле 2. Версия 3 реплицируется на узел 1. Там также выявляется причинно-следственная связь между версиями записи, и старые версии 1, 2 удаляются на узле 1. В результате в базе данных будет храниться одна версия записи (версия 3).

Рассмотренная схема «*master–master*» поддерживается в *NoSQL Riak* [1, 16], *Amazon Dynamo* [6], *Cassandra* [17] и др.

В некоторых случаях работа с вектором часов вызывает неудобства [5]. Потому базы данных *NoSQL* позволяют отключать вектор часов.

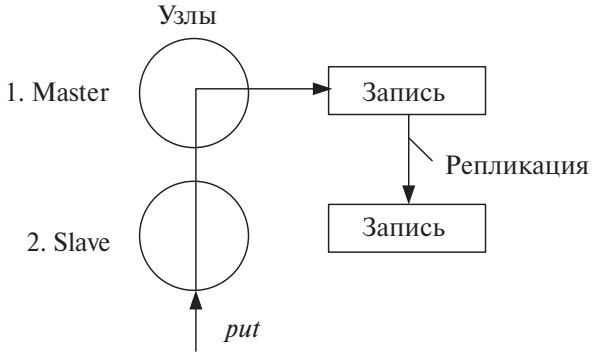

Рисунок 1. Репликация записей по схеме «master–slave»

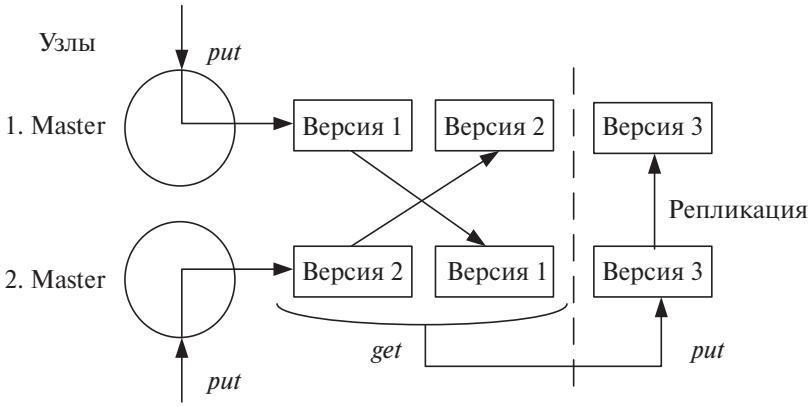

Рисунок 2. Репликация записей по схеме «master–master»

При возникновении конфликта обновления в базе данных сохраняется какая-то одна версия записи. Остальные конфликтующие версии сохраняются отдельно (по аналогии с *NoSQL CouchDB* [1, 18]). При чтении клиент получает запись из базы данных и все конфликтующие версии, после чего он решает, что с ними делать, то есть анализ причинно-следственных связей переносится на сторону клиента. Как будет показано далее, такой перенос не оказывает влияния на разработанную имитационную модель процесса ведения версий записи.

В работах [19–22] рассмотрены общие принципы построения вектора часов. Но они не учитывают важных особенностей применения векторов часов при анализе причинно-следственных связей между версиями записи.

В проанализированных публикациях отсутствуют упоминания о наличии формализованной схемы (алгоритма) формирования и анализа векторов часов, а также о разработке математических моделей процессов ведения версий записи в *NoSQL*.

Алгоритмы формирования вектора часов и анализа причинно-следственных связей

На рис. 3 показан пример ведения вектора часов (в скобках показан вектор часов записи). Запись D (например, какой-то документ) обновляется пользователями A_1, A_2, A_3 на одном узле. Первые два обновления выполняются пользователем A_1 последовательно. Далее пользователи A_2, A_3 одновременно читают и обновляют эту запись (случайное совпадение). В базе данных сохраняются две версии записи: D_1 и D_2. При чтении документа D пользователю A_1 возвращаются две версии записи с одним и тем же ключом (D_1 и D_2). Он вносит изменения – например, объединяет обновления, выполненные

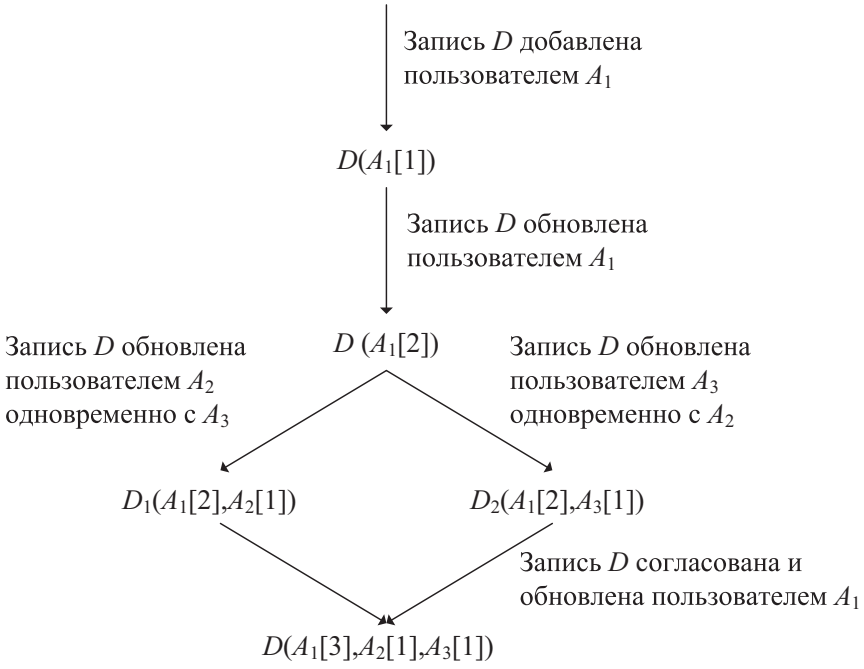

Рисунок 3. Пример ведения вектора часов

пользователями A_2, A_3. В базе сохраняется одна согласованная версия записи с вектором часов, включающим идентификаторы трех пользователей.

Преимущества вектора часов: отсутствие единой точки отказа системы; нет необходимости использовать временные метки, т.е. выполнять точную синхронизацию времени с одним эталоном.

Недостатки вектора часов: отсутствие возможности автоматически разрешать сложные конфликты; увеличение длины вектора часов при многократном обновлении записи. Однако в *NoSQL* существуют механизмы усечения вектора часов. Например, в системе *Riak* можно задавать частоту обрезания вектора на уровне сегмента, а также максимальный размер (длину) вектора часов [16].

Имитационная модель процесса ведения версий записи

Процесс обработки (согласования) клиентом версий записи представим в виде следующих шагов:

1. Приложение получает все версии с тем же ключом (момент τ_i) и предлагает их i-му клиенту в форме, удобной для анализа ($i = 1...K$).
2. Клиент анализирует эти версии и обрабатывает их согласно своей роли: объединяет мнения других пользователей (данный клиент играет роль руководителя группы), вносит свои предложения (клиент является рядовым членом группы) и т.д., назовем это случайное время анализа временем обработки (время φ).
3. Приложение сохраняет обновленную запись в базе (*NoSQL* формирует вектор часов, может быть, удаляет старые версии записи, добавляет в базу данных новую версию – обновление числа версий записи W).
4. Клиент через некоторое время возвращается к ранее откорректированной записи (документу), назовем это время временем обдумывания (случайное время, распределенное по экспоненциальному закону с параметром $1/\lambda$).
5. Перейти к шагу 1.

В соответствии с описанным выше процессом разработана модель ведения версий записи в базах данных *NoSQL* (рис. 4).

Модель включает следующие шаги, которые соответствуют процессу ведения версий записи (сами версии записей в модели не хранятся):

1. Занять позицию в массиве обработки версий записи (массив MS) в момент τ_i.
2. Рассчитать случайное время обработки φ и задержать процесс модели на это время.
3. Обновить массив MS.
4. Обновить число версий записи (W).
5. Занять позицию в источнике и задержать процесс модели на случайное время обдумывания.

Массив обработки MS имеет 2 столбца: Q_i – это число версий записи, которые необходимо удалить из базы данных в момент, когда i-й источник покидает фазу обработки; U_i – количество обновлений записи другими источниками (клиентами), выполненных между последовательными обновлениями записи данным клиентом.

Время реализации шагов 1, 3, 4 намного меньше времени выполнения шагов 2 и 5, поэтому это время в модели не учитывается.

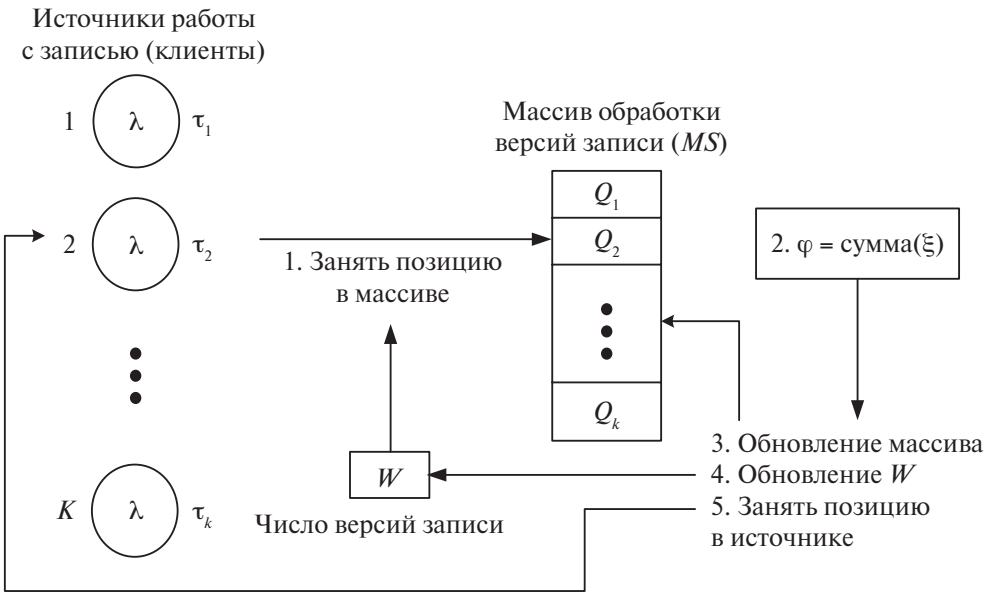

Рисунок 4. Модель ведения версий записи

Время φ на шаге 2 рассчитывается исходя из числа всех обновлений, выполненных другими клиентами между последовательными обновлениями записи данным клиентом.

Функцию распределения вероятностей числа версий записи (W), одновременно хранящихся в базе данных *NoSQL*, сложно представить в аналитическом виде. Поэтому была разработана имитационная модель на языке *GPSS* [7]. Ниже приведен алгоритм работы модели. Эта модель позволяет оценить средние значения и правые границы доверительных интервалов числа версий записи (W) и времени их обработки (φ).

Алгоритм работы имитационной модели

1. Положить $W = 1$, массив обработки MS пуст.
2. j-е требование поступает на обработку.
3. Сохранить текущее число версий записи в j-й строке массива MS: $Q_j = W$, определить φ, положить $U_i = 0$, задержать требование на время φ (обработать обновления записи).
4. Если j-е требование покидает массив обработки (j-я строка массива MS), то надо выполнить следующие действия:
   а) $W = W - Q_j + 1$;
   б) для всех строк массива MS, $i = 1...K$:

$$Q_i = Q_i - Q_j,$$

   если $Q_i < 0$, то положить $Q_i = 0$;
   в) если $i <> j$, то увеличить U_i на 1;
   г) задержать j-е требование на время обдумывания клиентом результатов работы с версиями.
5. Перейти к п. 2.

В таблице даны пояснения к пунктам алгоритма.

Как следует из описания модели и таблицы, здесь не требуется хранить образы версий записи. Это делает модель компактной и быстрой. Более того, разработанная модель инвариантна относительно места, где выполняется анализ причинно-следственных связей между версиями и их удаление (т.е. на сервере или на клиенте). Модель не изменяется, если в примечании к п. 4а алгоритма (см. таблицу) заменить удаление версий узлом на удаление их клиентом.

Ниже приведено описание подготовки натурных экспериментов для оценки адекватности разработанной модели процесса ведения версий записи.

Описание экспериментальной установки

На рынке существует много компаний, предоставляющих облачные вычислительные ресурсы. В наших экспериментах использовались виртуальные узлы (серверы), предоставленные компанией *DigitalOcean* (*DO*) [23], и *NoSQL Riak* [16]. Число узлов в кластере менялось от 5 до 10. На этих узлах запускались программы (процессы) клиентов (5 клиентов – на 5 узлах; 10, 15, 20 клиентов – на 10 узлах).

Таблица. Пояснение пунктов алгоритма

Пункт	Пояснения
1	Новая запись включена в БД
2	Наступил момент времени τ_j
3	j-й клиент читает W версий записи, $\varphi = \sum_{1}^{U_j+1} \xi$, ξ – случайное время обработки клиентом одного обновления записи, U_j – число обновлений, выполненных другими клиентами
4а	Клиент j обработал прочитанные версии записи (затратив на это время φ), сформировал вектор часов, процесс в узле удаляет из БД все старые версии записи (если их ранее не удалил другой процесс, т.е. если $Q_j \neq 0$), процесс добавляет в БД версию записи (+1)
4б	Учесть, что Q_j версий записи уже удалены из БД текущим j-м требованием и их не надо удалять i-м требованием (если $Q_j \neq 0$). Затем покидающее i-е требование удалит ранее им считанные версии, которые остались неудаленными, и увеличит число версий записи W на единицу (см. п. 4а – другие версии записи, поступившие позже чтения записи i-м клиентом, этот клиент удалить не может; эти версии будут содержать в векторе часов новые номера версий других клиентов – это объясняет, почему только минус Q_j в п. 4а).Дополнительно поясним п. 4б алгоритма. Пусть i-е требование появилось позже j-го. Если в этот момент $Q_j > 0$ (число еще не удаленных версий, которые ранее прочитал j-й клиент), то i-й клиент прочитает и эти версии (каждый клиент читает все версии записи, присутствующие в этот момент в базе данных). Аналогичные рассуждения можно провести для случая, если j-е требование появилось позже i-го, но было обработано раньше
4в	Накапливать для i-го клиента число обновлений, выполненных другими клиентами

При первоначальной настройке узла (*Droplet*) необходимо выбрать операционную систему (*ОС*) или снимок, ранее созданный пользователем. Использовалась ОС *Ubuntu Server* 14.04 [24], предустановленная поставщиком облачных услуг. Для установки и настройки *Riak* необходимо выполнить ряд действий на каждом из узлов. Установка базы данных *Riak* «с нуля» требует много времени. Поэтому для ускорения подготовки кластера общая часть действий по установке системы, описанных в [16], была выполнена один раз на одном узле. Далее был сделан снимок узла, который впоследствии восстанавливался на остальных узлах.

Подготовка и проведение экспериментов

В модели процесса ведения версий записи предлагалось использовать два столбца массива обработки *MS*. В первом столбце (*Q*) хранится число версий записи, которые необходимо удалить клиентом из базы данных, во втором (*U*) – число обновлений, выполненных другими клиентами между последовательными обновлениями записи данным клиентом. В соответствии с этим при проведении натурного эксперимента использовались две записи:

1. Основная запись *Z*; в БД могут храниться несколько версий этой записи.
2. Служебная (вспомогательная) запись *RZ*. Она содержит массив, аналогичный столбцу *U* модели обработки версий записи, где хранятся счетчики обновлений записи *Z*. Размер массива равен числу клиентов. При использовании строгой согласованности [1, 4, 25] клиент всегда получает актуальную запись. Но даже этот тип согласованности не может предотвратить конфликты, связанные с изменениями вспомогательной записи *RZ*. Это связано с тем, что при проведении натурного эксперимента запись *RZ* могут одновременно обновлять несколько клиентов, и текущий клиент получает несколько версий этой записи. Для разрешения таких конфликтов была разработана специальная программа.

На узлах запускались программы обработки версий записи *Z*, то есть эти программы выступали в роли клиентов. При проведении натурного эксперимента каждый клиент в цикле читает все версии записи *Z*, обрабатывает их и сохраняет уже одну согласованную версию этой записи в базе данных (результат обработки). Для расчета времени обработки версий записи *Z* клиент читает запись *RZ*, где хранится число обновлений записи *Z*, выполненных другими клиентами (это случайная величина). Время обработки (φ), число версий записи (*W*) и другие параметры фиксировались в журнале каждой программы. После проведения каждого эксперимента эти журналы обрабатывались, оценивались требуемые показатели (см. ниже), которые сравнивались с модельными значениями.

Среднее время обдумывания рассчитывалось по формуле $T\_A = 1/\lambda = 5{,}5\,r\,10$, где 5,5 – среднее условное (модельное) время обработки клиентом одного обновления записи (среднее значение ξ – см. таблицу); *r* – варьируемый параметр; 10 – значение одной условной единицы в мс при проведении натурных экспериментов. Одну модельную единицу проектировщик может трактовать по своему усмотрению. Это может быть одна или несколько секунд или минут, какая-то доля часа и т.д. В натурном эксперименте одна условная единица работы модели *GPSS* принималась равной 10 мс.

Анализ адекватности модели

Целью проведенных натурных экспериментов являлось доказательство адекватности разработанной модели процесса ведения версий записи.

Проводились четыре серии натурных экспериментов для числа клиентов $K = 5, 10, 15, 20$. В каждой серии изменялся параметр $r = 0{,}5, 0{,}8, 1{,}0, 3{,}0, 5{,}0$. Интервал времени каждого эксперимента рассчитывался так, чтобы обеспечить уровень надежности статистики 0,99 [26].

На рис. 5, 6 показаны зависимости средних значений (*VA*) и правых границ доверительного интервала (ПГДИ) (*VD*) числа *W* версий записей от *r* для различного числа клиентов *K*. ПГДИ оценивалась на уровне 0,99. Средняя относительная погрешность оценки характеристик случайного числа версий записи с помощью модели составляет: среднего значения – 7,5% и ПГДИ – 3,0%. Причем максимальная погрешность и по ПГДИ, и по среднему значению не превышает 14%.

Из рис. 5, 6 видно, что зависимости *VA* и *VD* от *r* имеют выраженный максимум (кроме $K = 5$). С увеличением времени между последовательными чтениями версий записи (времени обдумывания, т.е. с ростом *r*) среднее число версий записи *VA* стремится к 1 для каждого *K* (числа клиентов), но очень медленно. Так, при $r = 100$ $VA = 1, 2, 3$ соответственно для $K = 5, 10, 20$.

На рис. 7, 8 показаны зависимости средних значений (*TA*) и ПГДИ (*TD*) времени обработки φ клиентом обновлений записи от *r* для различного числа клиентов *K* (время пересчитано в условные модельные единицы). Средняя относительная погрешность оценки с помощью модели среднего значения времени обработки обновлений записи составляет 0,22%, ПГДИ – 5,6%.

Из графиков на рис. 7 видно, что среднее значение времени обработки обновлений (*TA*) практически постоянно для каждого *K*. Это можно объяснить так: за промежуток времени *t* между

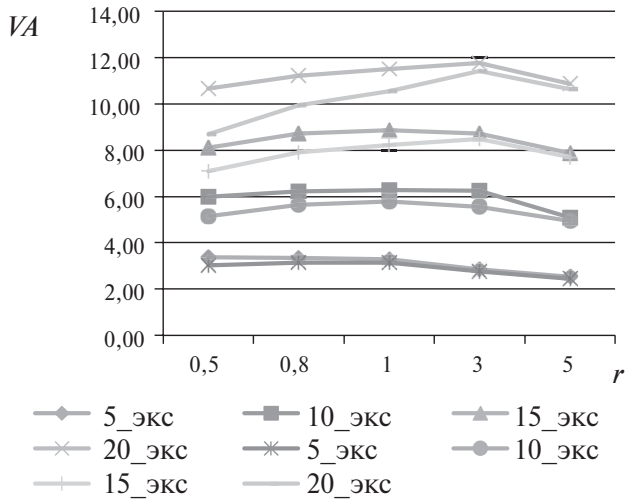

Рисунок 5. Среднее значение числа версий записей

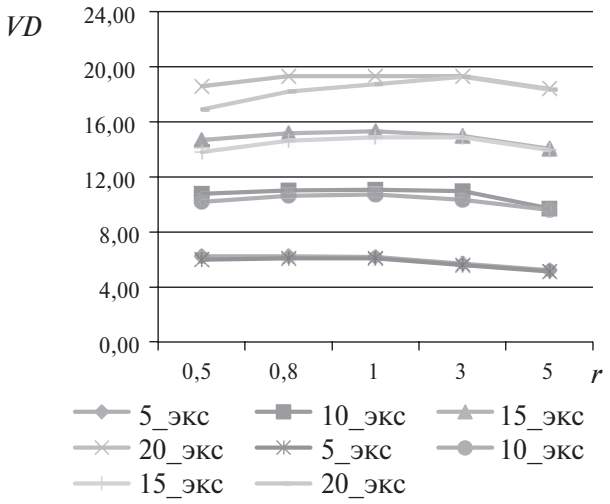

Рисунок 6. Правая граница доверительного интервала (ПГДИ) числа версий записей

последовательными чтениями версий записи клиентом в среднем будет выполнено $t[(K-1)/t]+1=K$ обновлений, что и объясняет отсутствие зависимости TA от r. ПГДИ времени обработки обновлений (уровень надежности равен 0,99) имеет минимум (см. рис. 8). То есть с ростом r TD сначала убывает, а затем возрастает. Так, $TD=400$ для $r=100$ и $K=20$ (при среднем значении $TA=110$). Это означает, что за один цикл работы какого-либо клиента (время между чтениями версий записи) другие клиенты могут обновить запись несколько раз.

Заключение

В результате проведенных исследований была разработана имитационная модель, позволяющая оценивать характеристики случайного числа версий записи и времени их обработки (времени согласования версий).

Для практического применения модели было разработано инструментальное средство. Для автоматической замены исходных данных в тексте модели (K, r и др.) разработана программа на языке PLUS [7]. Текст программы на PLUS и модели на GPSS генерируется инструментальным средством автоматически после ввода исходных данных.

Результаты натурных экспериментов показывают, что разработанная имитационная модель процесса ведения версий записи является адекватной (рис. 5–8). Средняя относительная погрешность по всем измерениям составила 4%. Эта модель

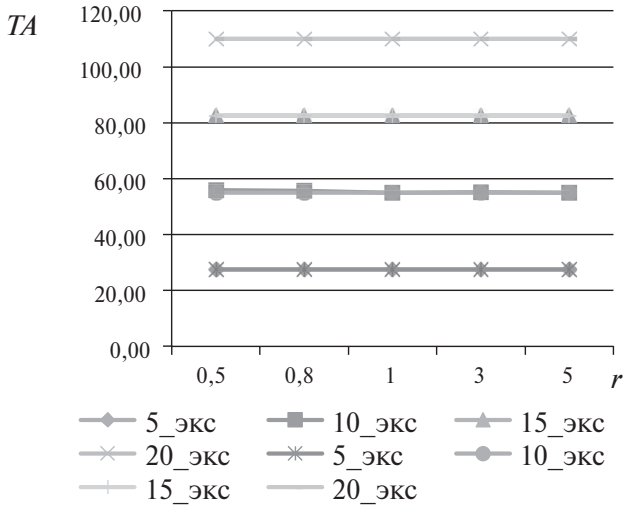

Рисунок 7. Среднее значение времени обработки клиентом обновлений записи

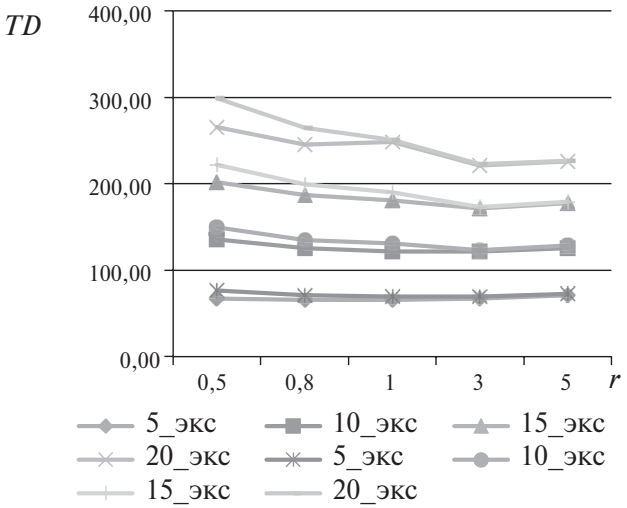

Рисунок 8. Правая граница доверительного интервала времени обработки клиентом обновлений записи

позволяет проверить, является ли число версий записи и/или время рассогласования версий допустимым для большого числа типов записей при экстремальной нагрузке на систему NoSQL. Это особенно важно на этапе проектирования, когда натурное моделирование крайне затруднено.

СПИСОК ЛИТЕРАТУРЫ

1. Redmond E., Wilson J.R. Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. Pragmatic Bookshelf, 2012.
2. Sadalage P.J., Fowler M. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education, 2012.
3. Hecht R., Jablonski S. NoSQL evaluation: A use case oriented survey. Cloud and Service Computing (CSC), 2011 International Conference on. IEEE, 2011, pp. 336–341.
4. Burdakov A. et al. Estimation models for NoSQL database consistency characteristics. Parallel, Distributed, and Network-Based Processing (PDP), 24th Euromicro International Conference on. IEEE, 2016, pp. 35–42.
5. Why Cassandra doesn't need vector clocks. Available at: http://www.datastax.com/dev/blog/why-cassandra-doesnt-need-vector-clocks (accessed 10.11.2017)
6. De Candia G. Dynamo: amazon's highly available key-value store. CM SIGOPS operating systems review, 2007, vol. 41, no. 6, pp. 205–220.
7. GPSS World reference manual. Minuteman Software. Fifth Edition. Holly Springs, NC, U.S.A., 2009.
8. Saito Y., Shapiro M. Optimistic replication. ACM Computing Surveys (CSUR), 2005, vol. 37, no. 1, pp. 42–81.
9. Chandra T.D., Toueg S. Unreliable failure detectors for reliable distributed systems. Journal of the ACM (JACM), 1996, vol. 43, no. 2, pp. 225–267.
10. Mongo DB. Available at: http://mongodb.org (accessed 10.11.2017)
11. Amazon Dynamo DB. Available at: http://aws.amazon.com/dynamodb (accessed 10.11.2017)
12. Redis. Available at: http://redis.io (accessed 10.11.2017)
13. Fowler M. Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc., 2002.
14. Strauch C. NoSQL databases. Lecture selected topics on software-technology ultra-large scale sites. Manuscript. Stuttgart Media University. 2011.
15. Orend K. Analysis and classification of NoSQL databases and evaluation of their ability to replace an object-relational Persistence Layer. Architecture. 2010, pp. 100.
16. Riak documentation. Available at: http://docs.basho.com/index.html (accessed 10.11.2017)
17. Cassandra. Available at: http://cassandra.apache.org (accessed 10.11.2017)
18. CouchDB. Available at: http://couchdb.apache.org (accessed 10.11.2017)
19. Tanenbaum A.S., Van Steen M. Distributed systems: principles and paradigms. Prentice-Hall, 2007.
20. Baldoni R., Klusch M. Fundamentals of distributed computing: A practical tour of vector clock systems. IEEE Distributed Systems Online, 2002, no. 2, pp. 1–25.
21. Sun C., Cai W. Capturing causality by compressed vector clock in real-time group editors. Parallel and Distributed Processing Symposium. Proceedings International, IPDPS2002, pp. 59–66.
22. Singhal M., Kshemkalyani A. An efficient implementation of vector clocks. Information Processing Letters, 1992. vol. 43, no. 1, pp. 47–52.
23. Digital Ocean. Available at: https://www.digitalocean.com (accessed 10.11.2017)
24. Ubuntu OS14.04. Available at: http://releases.ubuntu.com/14.04 (accessed 10.11.2017)
25. Bailis P. et al. Probabilistically bounded staleness for practical partial quorums. Proceedings of the VLDB Endowment, 2012. vol. 5, no. 8, pp. 776–787.
26. Zukerman M. Introduction to Queueing Theory and Stochastic Teletrac Models. Available at: http://www.ee.cityu.edu.hk/~zukerman/classnotes.pdf (accessed 10.11.2017)

REFERENCES

1. Redmond E., Wilson J.R. Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. Pragmatic Bookshelf, 2012.
2. Sadalage P.J., Fowler M. NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education, 2012.
3. Hecht R., Jablonski S. NoSQL evaluation: A use case oriented survey. *Cloud and Service Computing (CSC), 2011 International Conference on. IEEE*, 2011, pp. 336–341.
4. Burdakov A. et al. Estimation models for NoSQL database consistency characteristics. *Parallel, Distributed, and Network-Based Processing (PDP), 24th Euromicro International Conference on. IEEE*, 2016, pp. 35–42.
5. [Why Cassandra doesn't need vector clocks]. Available at: http://www.datastax.com/dev/blog/why-cassandra-doesnt-need-vector-clocks (accessed 10.11.2017)
6. De Candia G. Dynamo: amazon's highly available key-value store. *CM SIGOPS operating systems review,* 2007, vol. 41, no. 6, pp. 205–220.
7. GPSS World reference manual. Minuteman Software. Fifth Edition. Holly Springs, NC, U.S.A., 2009.
8. Saito Y., Shapiro M. Optimistic replication. *ACM Computing Surveys (CSUR),* 2005, vol. 37, no. 1, pp. 42–81.
9. Chandra T.D., Toueg S. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM),* 1996, vol. 43, no. 2, pp. 225–267.

10. [Mongo DB]. Available at: http://mongodb.org (accessed 10.11.2017)
11. [Amazon Dynamo DB]. Available at: http://aws.amazon.com/dynamodb (accessed 10.11.2017)
12. [Redis]. Available at: http://redis.io (accessed 10.11.2017)
13. Fowler M. Patterns of enterprise application architecture. Addison-Wesley Longman Publishing Co., Inc., 2002.
14. Strauch C. NoSQL databases. Lecture selected topics on software-technology ultra-large scale sites. Manuscript. Stuttgart Media University. 2011.
15. Orend K. Analysis and classification of NoSQL databases and evaluation of their ability to replace an object-relational Persistence Layer. Architecture. 2010, pp. 100.
16. [Riak documentation]. Available at: http://docs.basho.com/index.html (accessed 10.11.2017)
17. [Cassandra]. Available at: http://cassandra.apache.org (accessed 10.11.2017)
18. [CouchDB]. Available at: http://couchdb.apache.org (accessed 10.11.2017)
19. Tanenbaum A.S., Van Steen M. Distributed systems: principles and paradigms. Prentice-Hall, 2007.
20. Baldoni R., Klusch M. Fundamentals of distributed computing: A practical tour of vector clock systems. *IEEE Distributed Systems Online,* 2002, no. 2, pp. 1–25.
21. Sun C., Cai W. Capturing causality by compressed vector clock in real-time group editors. *Parallel and Distributed Processing Symposium. Proceedings International,* IPDPS2002, pp. 59–66.
22. Singhal M., Kshemkalyani A. An efficient implementation of vector clocks. *Information Processing Letters,* 1992. vol. 43, no. 1, pp. 47–52.
23. [Digital Ocean]. Available at: https://www.digitalocean.com (accessed 10.11.2017)
24. [Ubuntu OS14.04]. Available at: http://releases.ubuntu.com/14.04 (accessed 10.11.2017)
25. Bailis P. et al. Probabilistically bounded staleness for practical partial quorums. *Proceedings of the VLDB Endowment,* 2012, vol. 5, no. 8, pp. 776–787.
26. Zukerman M. [Introduction to Queueing Theory and Stochastic Teletrac Models]. Available at: http://www.ee.cityu.edu.hk/~zukerman/classnotes.pdf (accessed 10.11.2017)

ИНФОРМАЦИЯ ОБ АВТОРАХ

Григорьев Юрий Александрович, д.т.н., профессор, Московский государственный технический университет им. Н.Э. Баумана, 105005, Москва, ул. 2-я Бауманская, д. 5, стр. 1, тел.: 8 (916) 632-38-25, e-mail: grigorev@bmstu.ru.
Плутенко Андрей Долиевич, д.т.н., профессор, Амурский государственный университет, 675027, Амурская область, Благовещенск, Игнатьевское ш., д. 21, e-mail: plutenko@bk.ru.
Бурдаков Алексей Викторович, к.т.н., руководитель программ по разработке и внедрению автоматизированных информационных систем, компания «Информационные технологии для эпидемиологии», 109428, Москва, Рязанский пр-т, д. 10, стр. 2, e-mail: burdakov-bmstu@usa.net.
Цвященко Евгений Васильевич, к.т.н., ведущий специалист, отдел разработки программного обеспечения, ООО «Научно-производственная компания "Кронос-Информ"», 125130, Москва, ул. Приорова, д. 30, e-mail: tsviashchenko@gmail.com.

AUTHORS

Grigorev Yuriy, Dr., professor, Bauman Moscow State Technical University, 5/1, 2-ya Baumanskaya ulitsa, Moscow, 105005, Russian Federation, tel.: +7 (916) 632-38-25, e-mail: grigorev@bmstu.ru.
Plutenko Andrey, Dr., professor, Amur State University, 21, Ignatievskoe shosse, Amur Region, Blagoveshchensk, 675027, Russian Federation, e-mail: plutenko@bk.ru.
Burdakov Aleksey, PhD, head of programs on the development and implementation of automated information systems, «Information Technologies for Epidemiology», 10/2, Ryazanskiy prospekt, Moscow, 109428, Russian Federation, e-mail: burdakov-bmstu@usa.net.
Tsvyashchenko Evgeniy, PhD, leading specialist, Research and Production Enterprise Kronos-Inform, 30, ulitsa Priorova, Moscow, 125130, Russian Federation, e-mail: tsviashchenko@gmail.com.