Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 01. С. 128–136.

DOI: **10.7463/0115.0753706**

Представлена в редакцию: 08.01.2015

УДК 004.657

Анализ процессов обработки версий записи в базах данных NoSQL

профессор, д.т.н. Григорьев Ю. А.[1], **Цвященко Е. В.**[1,*]

*eugene.tsviashchenko@gmail.com

[1]МГТУ им. Н.Э. Баумана, Москва, Россия

В статье рассмотрены процессы обработки версий записей. Представлен алгоритм ведения версий записей в базах данных NoSQL. Разработана модель обработки версий записи при одновременной работе пользователей с записью. Модель позволяет оценить время согласования, а также число версий записи, одновременно существующих в базе данных. Представлено два варианта модели. В первом варианте время обработки пользователем версий записи рассчитывается из числа версий, считанных пользователем. Во втором варианте время обработки рассчитывается из числа обновлений, выполненных другими пользователями между последовательными обновлениями записи текущим клиентом. Представлен анализ модельных экспериментов.

Ключевые слова: база данных NoSQL, версии записи, вектор часов, модель согласования, доверительный интервал

Введение

Для повышения производительности и отказоустойчивости автоматизированных информационных систем (АИС) в настоящее время все чаще используются системы баз данных (БД), построенные на парадигме распределенных хранилищ «ключей/значений», получивших название NoSQL (Not-Only-SQL) [1]. В базах данных NoSQL не поддерживается режим ведения транзакций, поэтому возникает проблема согласования данных. Поддержание требуемого уровня согласованности для каждой конкретной предметной области может регулироваться параметрами (N, W, R) [2].

При слабой согласованности, когда $W+R \leq N$, пользователи могут одновременно обновлять записи с одним и тем же ключом, следовательно, система будет хранить несколько версий данной записи. В этом случае возникает проблема согласования версий (конфликт обновления). Базы данных NoSQL поддерживают механизм ведения вектора часов (Vector Clock - VC) [2] для каждой версии записи, хранящейся в БД, который содержит информацию о пользователях, выполнявших изменения данной версии записи.

При чтении пользователь получает все версии записи с данным ключом, обрабатывает их и сохраняет новую версию записи, при этом старые версии удаляются из хранилища.

При увеличении числа версий записи возрастает время их согласования клиентами (просмотр и объединение). В статье рассматривается механизм ведения вектора часов, предлагается имитационная модель обновления какой-либо записи, параллельно обрабатываемой пользователями. Рассматриваются два варианта модели, в которых по-разному рассчитывается время обработки пользователем считанных записей. Исследуется время согласования версий записи и число версий, одновременно хранящихся в хранилище NoSQL. Приводится анализ полученных результатов.

1. Ведение версий записи и вектора часов в NoSQL системах

При наличии нескольких узлов возможны конфликты данных, т.к. многие клиенты могут одновременно обновлять записи (документы), имеющие одинаковые ключи и хранящиеся в разных репликах. Конечно, подобные конфликты можно разрешить, если каждую запись базы данных снабдить временной меткой и отдавать предпочтение той версии записи, у которой метка самая последняя. Однако в кластере узлов это решение будет работать, если все часы точно синхронизированы, что часто является невыполнимой задачей.

СУБД NoSQL типа Riak [3] предоставляет механизм, позволяющий разрешать конфликты, используя так называемые векторные часы. Векторные часы – это последовательность пар <пользователь, номер версии записи для этого пользователя>, которая описывает порядок обновления этой записи. Кратко рассмотрим алгоритмы формирования векторных часов и обновления записей [4].

Пусть $\{A_i\}$ – множество идентификаторов пользователей (клиентов), обновляющих записи базы данных. Рассмотрим два варианта.

Случай 1. Пользователь $A_m \in \{A_i\}$ добавляет новую запись. Для этой записи СУБД установит следующий вектор часов: $VC = A_m[1]$, где 1 – номер версии записи для пользователя A_m.

Случай 2. Пользователь A_m читает запись по ключу, а затем обновляет её. Ниже приведены алгоритмы формирования вектора часов и обновления записи в базе данных для этого случая.

1. Алгоритм 1 формирования вектора часов VC новой версии записи.

   Вход: A_m – идентификатор пользователя, который читает запись. $\{VC_n\}$, $VC_n = \{Ai[j]\}_n$ - вектор часов прочитанной пользователем A_m n-ой версии записи (при чтении пользователь A_m получает все версии записи с тем же ключом).

   Алгоритм:

   К = 0; VC = ∅;

ДЛЯ каждой прочитанной n-ой версии записи

ЕСЛИ $A_m[j_n] \in VC_n$, ТО $VC = VC \cup (VC_n - A_m[j_n])$;

$K = \max(K, j_n)$;

КОНЕЦ ДЛЯ

$VC = VC \cup A_m[K + 1])$.

2. Алгоритм 2 обновления записи в базе данных.

   Вход: $\{VC_n\}$, $VC_n = \{A_i[j]\}_n$ - вектор часов существующей в базе данных n-ой версии записи (не обязательно прочитанной пользователем). $VC = \{A_i[j]\}$ – сформированный алгоритмом 1 вектор часов новой версии записи.

   Алгоритм:

   ДЛЯ каждой существующей в базе данных n-ой версии записи

   ЕСЛИ $\forall M$: $A_M[L] \in VC_n$ ($A_M[K] \in VC$ и $L \leq K$), ТО удалить существующую n-ую версию записи из базы данных;

   КОНЕЦ ДЛЯ

   Включить в базу данных новую запись с вектором часов VC и с тем же ключом.

Из последнего алгоритма следует, что для записи с одним и тем же ключом в базе данных могут одновременно храниться несколько версий записей. В этом случае при чтении записи по этому ключу пользователь получает все версии записи. На рис. 1 представлен пример обновления записи с использованием вектора часов.

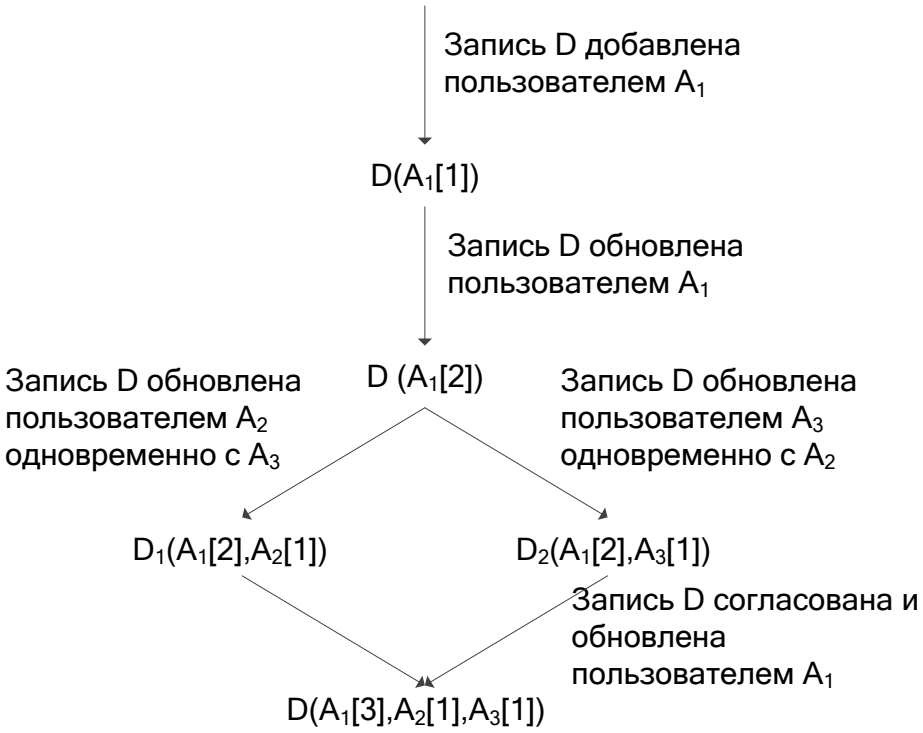


Рис. 1. Пример использования вектора часов

Запись D (например, какой-то документ) обновляется пользователями A_1, A_2, A_3. В скобках показан вектор часов записи. Первые два обновления выполняются пользователем A_1 последовательно. Далее пользователи A_2, A_3 одновременно 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 клиентами A_2, A_3, или добавляет новые обновления. В базе сохраняется одна версия записи с вектором часов, включающим идентификаторы трёх пользователей.

С течением времени длина вектора часов растёт. В NoSQL существуют механизмы, позволяющие его «обрезать» [2]. При увеличении числа версий записи, одновременно хранящихся в базе данных, возрастает время их согласования клиентами (просмотр, обновление или объединение). В статье разработана модель оценки времени работы какого-либо клиента с этими версиями.

Распределение числа версий записи, одновременно хранящихся в NoSQL, сложно представить в аналитическом виде. Поэтому в работе используется имитационный подход к моделированию. Оцениваются характеристики случайного числа версий записи и времени их обработки пользователем.

2. Модель обновления версий записи (вариант 1)

Представим процесс обработки клиентом версий записи в виде следующих шагов:

1. Приложение получает все версии с тем же ключом (читает пары «ключ/значение») и предлагает их клиенту в форме, удобной для анализа.
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.; назовём это время *временем обработки*.
3. Приложение сохраняет обновлённую запись в базе (NoSQL формирует вектор часов, может быть удаляет старые версии записи, добавляет в базу данных новую версию).
4. Клиент через некоторое время возвращается к ранее откорректированной записи (документу); назовём это время *временем обдумывания*. Перейти к шагу 1.

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

На рис. 2 представлена схема работы модели.

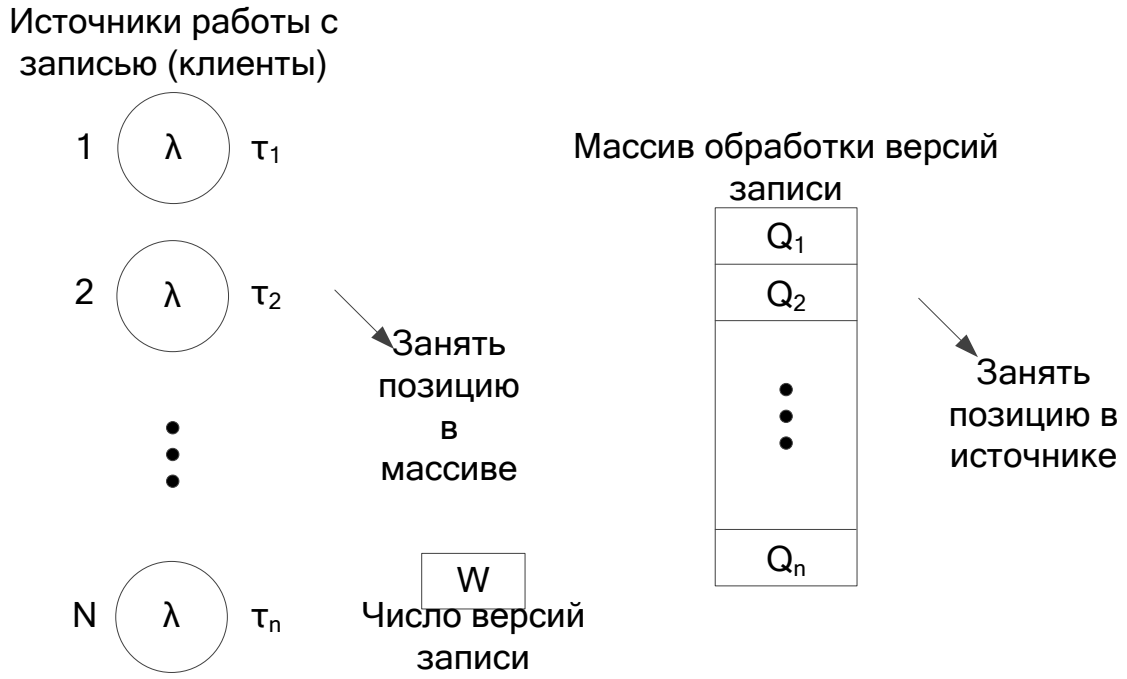


Рис. 2. Модель обработки версий записи

На рис. 2 приняты следующие обозначения:

λ – параметр экспоненциального закона распределения времени обдумывания клиентом (1…N) результатов работы с версиями записи,

τ_i – момент поступления требования на чтение версий записи от i-го источника (клиента): i-й клиент читает W версий записей, $\varphi = \sum_{1}^{W} \xi$ - время обработки всех версий записи клиентом, ξ - случайное время обработки, связанное с одной версией записи,

W – текущее число версий записи в базе данных NoSQL.

Массив обработки версий записи содержит один столбец. Значение Q_i – это число версий записи, которые необходимо удалить из базы данных в момент, когда i-й источник покидает фазу обработки.

В табл. 1 приводится описание алгоритма работы модели.

Описанный в табл. 1 алгоритм был реализован в среде GPSS [5]. GPSS является средством моделирования параллельных процессов. Система позволяет задавать различные распределения вероятностей для случайного времени пребывания в источнике

(времени обдумывания) и времени обработки клиентом одной версии записи, а также имеет встроенные механизмы сбора статистики и построения гистограмм.

Таблица 1. Алгоритм модели по варианту 1

№	Пункт алгоритма	Примечание
1	Положить W=1, массив обработки пуст.	Новая запись включена в БД.
2	j-ое требование поступает на обработку.	Наступил момент времени τ_j.
3	Сохранить текущее число версий записи в j-ой строке массива: $Q_j = W$, определить φ, задержать транзакт (требование) на время φ (обработать записи).	j-й клиент читает W версий записи из базы данных, $\varphi = \sum_{1}^{W} \xi$, ξ - случайное время обработки, связанное с одной версией записи.
4	Если очередное j-ое требование поступает на обработку, то перейти к пункту 3 алгоритма	Наступило очередное время τ_j
5	Если требование покидает массив обработки (пусть это будет j-я строка массива), то выполнить следующие действия: а) $W: = W - Q_j + 1$ б) для всех строк массива обработки $Q_i := Q_i - Q_j$, если $Q_i < 0$, то положить $Q_i = 0$ в) задержать клиента на время обдумывания результатов работы с версиями.	Клиент j обработал прочитанные версии записи (затратив на это время φ_j), сформировал вектор часов, удаляет из БД все старые версии записи (если их ранее не удалил другой клиент, т.е. если $Q_j \neq 0$), добавляет в БД свою версию записи (+1, см. алгоритм 2 ведения версий записи). Учесть, что Q_j, версий записи уже удалены из БД текущим j-м клиентом и их не надо удалять i-м клиентом (если $Q_j \neq 0$). Затем покидающее i-ое требование удалит ранее им считанные версии, которые остались не удалёнными, и увеличит число версий записей W на единицу (см. пункт 5а – другие версии записей, поступившие позже чтения записей i-м клиентом, этот клиент удалить не может, т.к. они будут содержать в векторе часов новые номера версий других клиентов – это объясняет, почему только минус Q_j в 5а).
6	Перейти к пункту 4 алгоритма	

В качестве функции распределения вероятностей времени ξ обработки клиентом одной версии записи использовалась функция, обратная следующей функции GPSS:

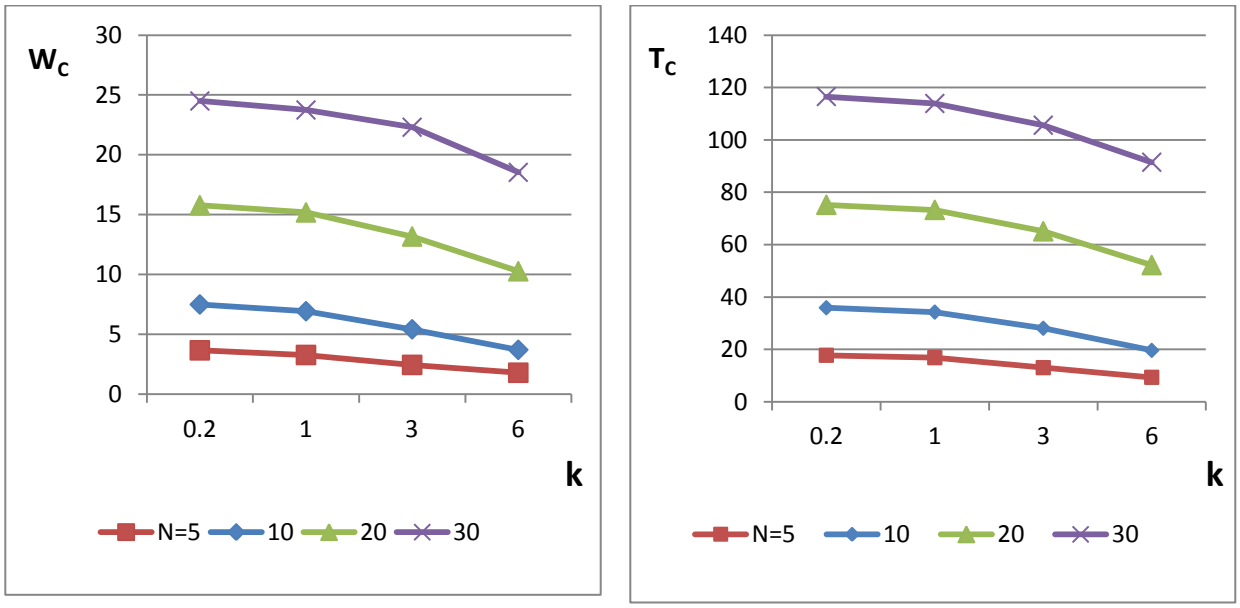
VRF FUNCTION RN1,D7

0,0/.125,1/.375,2/.5,3/.625,8/.875,9/1.0,10 (1)

Плотность соответствующей функции распределения является двугорбой со средним значением, равным 5.5.

$1/\lambda$ – среднее время обдумывания (т.е. среднее время между последовательными чтениями версий записи) принималось равным величие $k*t_0$, где t_0 – среднее время обработки клиентом одной версии записи (t_0=5.5), k = 0.2; 1; 3; 6.

На рис. 3 представлены результаты моделирования: средние значения числа версий записи (W_C) и времени обработки документа (версий записи) одним клиентом (T_C), N – число клиентов. Единицей измерения времени T_C (ось ординат) может быть секунда, минута, какая-то доля часа и т.д.



а) б)

Рис. 3. Среднее значение числа версий записи WC (а) и времени обработки версий записи клиентом TC (б)

Можно заметить, что с увеличением среднего времени обдумывания (т.е. с ростом k) среднее число версий записи уменьшается (рис. 3а) и стремится к 1 для каждого N. Причем, уменьшается тем быстрее, чем больше число клиентов, одновременно обрабатывающих запись. Соответственно уменьшается и среднее время обработки версий записи (рис. 3б). Ясно, что при этом функция распределения вероятностей времени обработки этих записей стремится к закону, определённому функцией (1).

Рассмотренная модель не учитывает в явном виде того, что время обработки версий записи зависит от числа обновлений, выполненных другими клиентами между последовательными обновлениями записи данным клиентом. Поэтому предлагается второй вариант модели обработки версий записи.

3. Модель обработки версий записи (вариант 2)

Модель отличается от предложенной ранее тем, что в массив обработки добавляется столбец ($U_1 \ldots U_n$), который содержит количество обновлений записи другими источниками (клиентами), выполненных между последовательными обновлениями записи данным клиентом. Время обработки рассчитывается не как сумма по W, а как сумма по U_i.

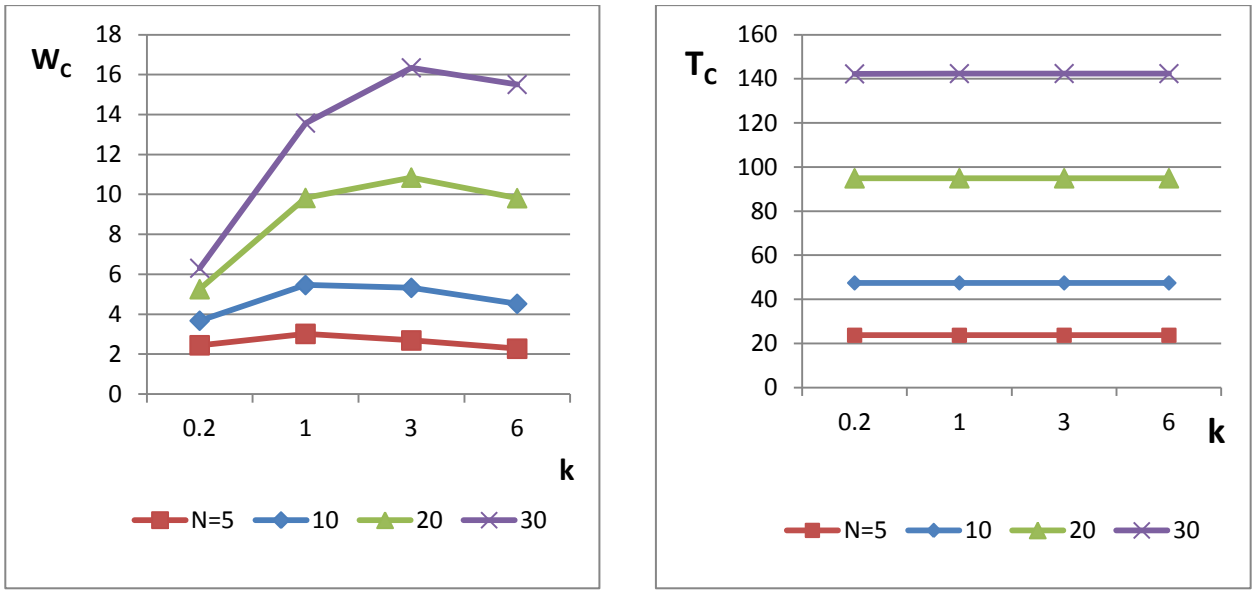
В таблице 2 приводится описание алгоритма работы модели.

Таблица 2. Алгоритм модели по варианту 2

№	Пункт алгоритма	Примечание
1	Положить W = 1, массив обработки пуст	Новая запись включена в БД.
2	j-ое требование поступает на обработку	Наступил момент времени τ_j
3	Сохранить текущее число версий записи в j-ой строке массива: $Q_j = W$, определить φ, положить $U_i=0$, задержать требование на время φ (обработать обновления записи).	j-й клиент читает W версий записей, $\varphi = \sum_{1}^{U_j+1} \xi$, ξ - случайное время обработки клиентом одного обновления записи, U_i - число обновлений, выполненных другими клиентами.
4	Если j-ое требование поступает на обработку, то перейти к пункту 3 алгоритма.	Наступило очередное время τ_j
5	Если требование покидает массив обработки (пусть это будет j-я строка массива), то выполнить следующие действия: а) $W = W - Q_j + 1$ б) для всех строк массива $Q_i = Q_i - Q_j$, если $Q_i < 0$, то положить $Q_i = 0$, если i <> j, то увеличить U_i на 1 г) задержать клиента на время обдумывания результатов работы с версиями.	См. примечание в таблице 1. См. примечание в таблице 1. Накапливать для i-го клиента число обновлений, выполненных другими клиентами.
6	Перейти к пункту 4 алгоритма	

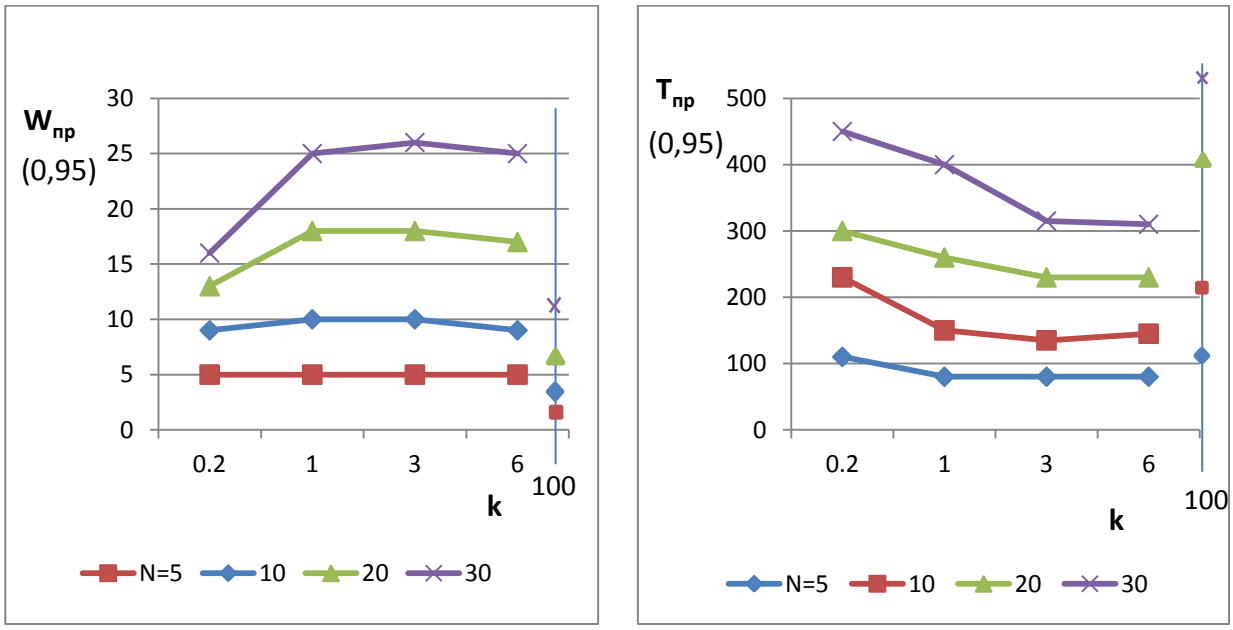
Исходные данные модельных экспериментов совпадают с данными для варианта 1. В качестве функции распределения вероятностей времени ξ обработки клиентом одного обновления, выполненного другим клиентом, использовалась функция, обратная (1).

Результаты моделирования представлены на рис. 4 и рис. 5.



а) б)

Рис. 4. Среднее значение числа версий записи W_C (а) и времени обработки версий записи клиентом T_C (б)



а) б)

Рис. 5. Правая граница доверительного интервала (уровень надёжности 0,95) числа версий записи $W_{пр}$ (а) и времени обработки версий записи клиентом $T_{пр}$ (б)

Неожиданный результат - зависимость W_C от «k» имеет выраженный максимум (рис. 4а). С увеличением времени между последовательными чтениями версий записи (времени обдумывания, т.е. с ростом «k») среднее число версий записи стремится к 1 для каждого N, как и в варианте 1, но очень медленно. Так при K=100 W_C=1, 2, 3, 4 соответственно для N= 5, 10, 20, 30. Из графика видно, что среднее значение времени обработки версий записи (T_C) (рис. 4б) практически постоянно для каждого N. Это можно объяснить так. За промежуток времени t между последовательными чтениями версий записи клиентом в среднем будет выполнено t(N-1)/t=N-1 обновлений, что и объясняет отсутствие зависимости T_C от «k». Причём чем выше N, тем выше корреляция между количеством обновлений, выполненных за время обдумывания, для разных клиентов. Для N=30 T_C= 142.4, а не (30-1)(t_0=5.5)= 160, как следовало ожидать. Эта тенденция сохраняется с ростом k. Так при k=100 T_C=23.7, 47.5, 95.0, 142.5 для N= 5, 10, 20, 30.

Правая граница доверительного интервала времени обработки версий записи (уровень надёжности равен 0.95) имеет минимум (рис. 5б) – это тоже любопытный результат.

С ростом «k» $T_{пр}$ сначала убывает, а затем возрастает (см. вертикальную линию справа для k=100). Так $T_{пр}$ =520 для k=100 и N=30. Это означает, что за один цикл работы какого-либо клиента (время между чтениями версий записи + время их обработки) другой клиент(ы) может обновить запись несколько раз.

Заключение

1. Разработана имитационная модель анализа времени обработки версий записи. Модель предусматривает параллельную обработку клиентами записи с одним и тем же ключом, хранящейся в распределенной системе NoSQL. Предложены два варианта модели. Согласно первому варианту, время обработки клиентом версий записи зависит от количества версий, считанных из базы данных NoSQL. Согласно второму варианту, время обработки клиентом версий записи зависит от числа выполненных обновлений записи другими клиентами между последовательными чтениями версий записи данным клиентом.

2. Выполнены модельные эксперименты в среде GPSS. Анализ результатов показал, что несмотря на внешнее сходство вариантов модели, характер изменения средних значений числа версий записи и времени их обработки существенно различается.

3. Во 2-м варианте зависимости среднего числа версий записей в базе данных и его правой границы доверительного интервала (W_C и $W_{ПР}$) от среднего времени обдумывания (т.е. от «k») имеют выраженный максимум (рис. 4а и 5а).

4. В этом же 2-м варианте среднее время обработки версий записи практически не зависит от среднего времени обдумывания (см. рисунок 4б). Но правая граница доверительного интервала этого времени имеет выраженный минимум (рис. 5б).

Список литературы

1. NoSQL. Википедия: Свободная энциклопедия. Режим доступа: http://ru.wikipedia.org/wiki/NoSQL (дата обращения 07.01.2015).
2. Редмон Э., Уилсон Д.Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. М.: ДМК Пресс, 2013. 384 с.
3. Riak documentation. Режим доступа: http://docs.basho.com/index.html (дата обращения 07.01.2015).
4. Григорьев Ю.А. Анализ свойств баз данных NoSQL // Информатика и системы управления. 2013. № 2. С. 3-13.
5. GPSS World Reference Manual // Minuteman Software: website. Режим доступа: http://www.minutemansoftware.com/reference/reference_manual.htm (дата обращения 07.01.2015).

Science & Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

Science and Education of the Bauman MSTU, 2015, no. 01, pp. 176–188.

DOI: **10.7463/0115.0753706**

Received: 08.01.2015

Analysis of Handling Processes of Record Versions in NoSQL Databases

Yu.A.Grigorev[1], E.V. Tsviashchenko[1,*] *eugene.tsviashchenko@gmail.com

[1]Bauman Moscow State Technical University, Moscow, Russia

Keywords: NoSQL database, record versions, vector clock, consistency model, confidence interval

This article investigates the handling processes versions of a record in NoSQL databases. The goal of this work is to develop a model, which enables users both to handle record versions and work with a record simultaneously. This model allows us to estimate both a time distribution for users to handle record versions and a distribution of the count of record versions. With eventual consistency (W=R=1) there is a possibility for several users to update any record simultaneously. In this case, several versions of records with the same key will be stored in database. When reading, the user obtains all versions, handles them, and saves a new version, while older versions are deleted. According to the model, the user's time for handling the record versions consists of two parts: random handling time of each version and random deliberation time for handling a result. Record saving time and records deleting time are much less than handling time, so, they are ignored in the model. The paper offers two model variants. According to the first variant, client's handling time of one record version is calculated as the sum of random handling times of one version based on the count of record versions. This variant ignores explicitly the fact that handling time of record versions may depend on the update count, performed by the other users between the sequential updates of the record by the current client. So there is the second variant, which takes this feature into consideration. The developed models were implemented in the GPSS environment. The model experiments with different counts of clients and different ratio between one record handling time and results deliberation time were conducted. The analysis showed that despite the resemblance of model variants, a difference in change nature between average values of record versions count and handling time is significant. In the second variant dependences of the average count of record versions in database and its right bound of the confidence interval on the average deliberation time have a pronounced maximum. In the same variant an average handling time of record versions practically does not depend on the average deliberation time, but the right bound of the confidence interval of this time has a pronounced minimum.

References

1. NoSQL. Wikipedia, the free encyclopedia. Available at: http://en.wikipedia.org/wiki/NoSQL , accessed 07.01.2015.
2. Redmond E., Wilson J.R. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2012. (Russ. ed.: Redmond E., Wilson J.R. *Sem' baz dannykh za sem' nedel'. Vvedenie v sovremennye bazy dannykh i ideologiyu NoSQL*. Moscow, DMK Press, 2013. 384 p.).
3. Riak Docs: Riak documentation. Available at: http://docs.basho.com/index.html , accessed 07.01.2015.
4. Grigor'ev Yu.A. Analysis of NoSQL database properties. *Informatika i sistemy upravleniya = Information Science and Control Systems*, 2013, no. 2, pp. 3-13.
5. GPSS World Reference Manual. Minuteman Software: website. Available at: http://www.minutemansoftware.com/reference/reference_manual.htm , accessed 07.01.2015.