Базы данных временных рядов и средства обработки

Намиот Д.Е.,
Московский государственный университет имени М.В. Ломоносова
dnamiot@gmail.com

Аннотация. В статье рассматриваются вопросы, связанные с хранением временных рядов. Временные ряды достаточно широко используются в различных приложениях. Типичным примером является последовательность измерений, которая выполняется на некотором временном интервале. Большей частью, процесс обработки будет связан с необходимостью хранения полученных данных. Соответственно, тема хранения временных рядов является весьма актуальной. Естественно, при определенных условия, которые обсуждаются в этой статье, можно использовать традиционные реляционные базы данных. Возможные ограничения будут, в первую очередь, связаны со скоростью поступления новых данных. С другой стороны, возможные ограничения, а также особенности обработки временных рядов приводят к необходимости использовать специализированные системы для хранения временных рядов. Именно описанию работы с временными рядами с точки зрения баз данных и посвящена эта статья. В ней рассматриваются вопросы использования традиционных реляционных баз данных для хранения временных рядов, а также решения, классифицируемые как NoSQL системы. Целью работы является описание процесса выбора решения по хранению и обработке данных для систем межмашинного взаимодействия (M2M) и Интернета вещей (IoT).

Ключевые слова: временные ряды, хранение данных, SQL, NoSQL

Введение

Классически, временной ряд представляет собой последовательность данных, обычно соответствующих каким-либо измерениям на некотором временном интервале. Подобного рода модели встречаются в самых разных прикладных областях. Самый простой пример – это системы для межмашинного взаимодействия (M2M) и Интернет вещей (Internet of Things – IoT). Любые сенсорные измерения и формируют временной ряд. В силу общности, задачи обработки временных рядов являются одной из традиционных (а равно – и одной из старейших) задач для баз данных.

Рассмотрим традиционные реляционные базы данных. На первый взгляд, проблема хранения временных рядов выглядит просто. Мы можем создать таблицу, указав столбец со временем в качестве ключа. Остальные столбцы таблицы будут представлять измеренные в определенный момент времени значения. Каждое новое измерение просто добавляет строку к таблице. Одно измерение (один отсчет) – одна строка. Естественно, в любой практической системе количество измеряемых значений конечно. Каждый сенсор в IoT системе имеет определенное число измеряемых значений. Но, на практике, в заданный момент времени мы будем иметь, скорее всего, только часть из потенциально возможных атрибутов. Устройства в системе, чаще всего, асинхронны, работаю по собственным циклам, со своими временными интервалами. Это означает, что каждая строка в таблице будет содержать только часть из потенциально возможных атрибутов. Остальные атрибуты будут пустыми (null value). Это ведет к неэффективному расходованию дискового пространства и затрудняет последующую обработку.

Если мы обратимся к операциям с базой данных для временных рядов, то очевидно, что операция добавления (INSERT) будет превалировать. Обновление и удаление – нетипичные операции для временных рядов. Но самый интересный момент заключается в том, что и операция чтения (SELECT) имеет свою специфику. Операция чтения напрямую связана с

методами обработки. Обработка непрерывно поступающих данных всегда связана с некоторым “окном” – частью данных за некоторый короткий интервал времени. Для примера рассмотрим популярный алгоритм Dynamic Time Warping (DTW) [1]. DTW описывает наложение двух сигналов, которое минимизирует расстояние между ними. Предположим, что у нас есть два временных ряда: *P* длиной *n* и *Q* длиной *m*:

$$P = p_1, p_2, \dots p_n \quad (1)$$

$$Q = q_1, q_2, \dots q_m \quad (2)$$

Для выравнивания двух последовательностей алгоритм конструирует матрицу $n \; X \; m$, где элемент на позиции (*i,j*) соответствует расстоянию:

$$d(p_i, q_j) = (p_i - q_j)^2 \quad (3)$$

Эта формула описывает расстояние между двумя точками временных рядов. DTW ищет лучшее соответствие между двумя последовательностями, минимизируя расстояние:

$$\text{DTW}(P, Q) = \min \sqrt{\sum_{k-1}^{k} w_k} \quad (4)$$

где W – так называемый путь трансформации. Это набор смежных элементов матрицы, который устанавливает соответствие между P и Q. Он представляет собой путь, который минимизирует общее расстояние между P и Q.

Мы представили это описание, чтобы подчеркнуть, что обрабатывается всегда некоторая часть ряда. DTW ищет соответствие между двумя фрагментами, а не между всеми последовательностями. Возвращаясь к базам данных это означает, что для баз данных временных рядов запросы чтения со сложными условиями не являются характерными. Необходимо просто читать очередную порцию данных (последовательно). Да, могут быть задачи типа биллинга, где придется обрабатывать весь ряд, но даже тогда это будет, большей частью прямое последовательное чтение. Проводя аналогии с оператором SELECT в языке SQL, реализация конструкции SELECT TOP N для выборки последних N записей важнее, чем тот же SELECT с условием. Обработка временных рядов, с точки зрения работы с данными, представляет

собой модельный пример для так-называемой лямбда-архитектуры [2] (рисунок 1).

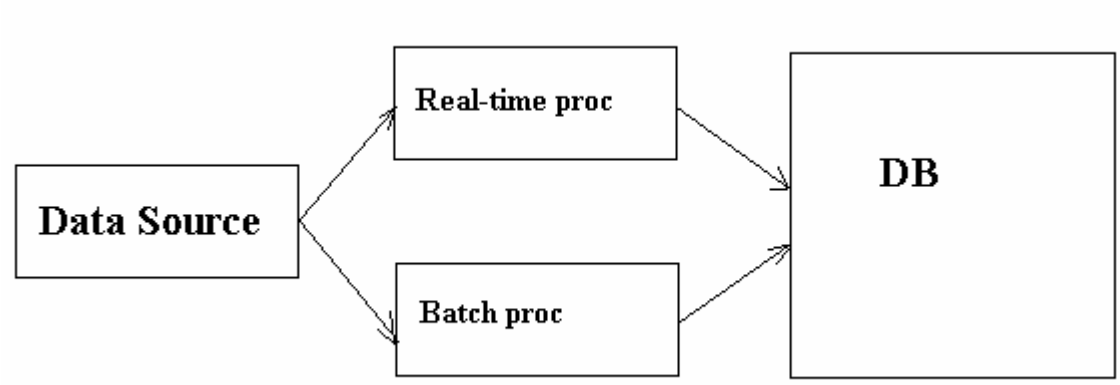

Рис.1 Лямбда-архитектура

Обработка данных разделяется на два потока – обработка поступающих данных в реальном времени и пакетная обработка. Большая часть поступающих данных обрабатывается именно в реальном времени. Это особенно справедливо для Интернета вещей (IoT) и межмашинного взаимодействия [3,4]. Выработка реакций на сенсорные измерения (что связано, большей частью, именно с обработкой временных рядов) относится к задачам реального времени. В то же самое время, ничто не препятствует существованию отдельной ветви для задач без строгих ограничений по времени.

Оставшаяся часть статьи структурирована следующим образом. В разделе 2 описываются решения по обработке временных рядов в реляционных базах данных. Раздел 3 посвящен NoSQL решениям.

Временные ряды в реляционных базах данных

В этом разделе мы хотим остановиться на работе с временными рядами в реляционных база данных. В качестве первого шага необходимо упомянуть систему TokuDB [5]. Мы рассматривали этот вариант как механизм работы с данными в MySQL. TokuDB использует фрактальные деревья в качестве

реализации индекса (в отличии от более привычных B-деревьев). Основное отличие состоит в том, что во фрактальных деревьях каждый узел содержит буфер для сохранения изменений. Соответственно, изменения сохраняются в промежуточных узлах, а далее сохраняются на диск уже большими блоками по мере накопления [6]. Естественно, что это влияет, в первую очередь, на скорость выполнения операции добавления данных. Именно там происходят массивные модификации, для сохранения которых требуется меньше дисковых операций. Разница проиллюстрирована на рисунке 2.

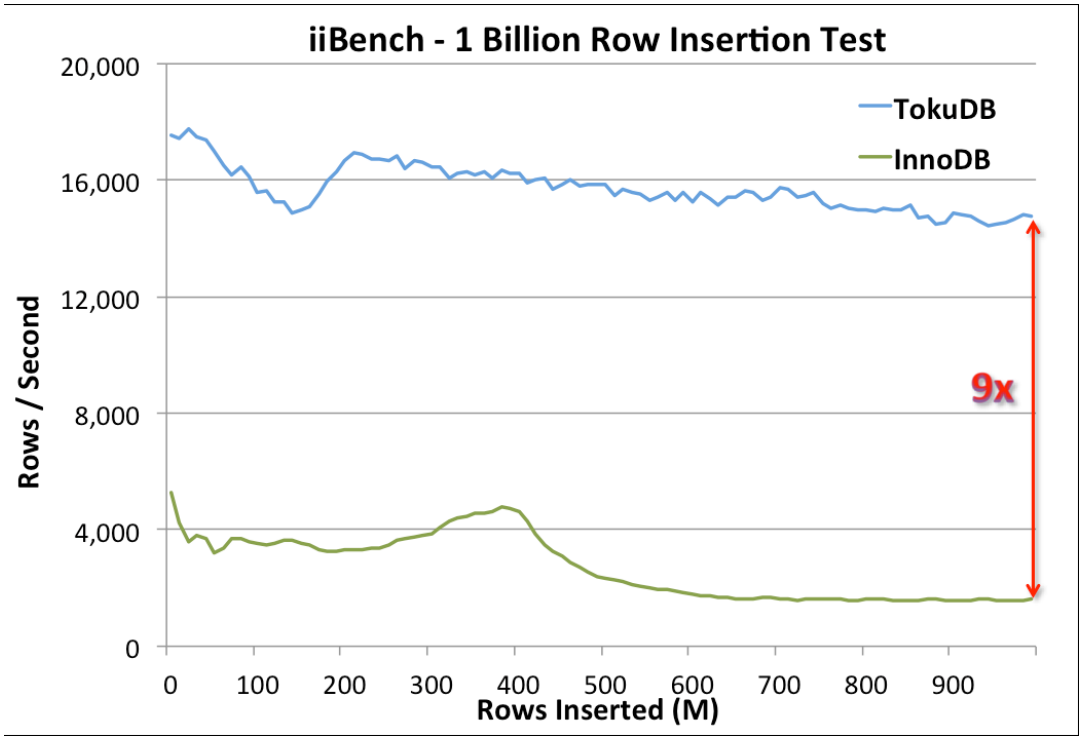

Рис. 2 Производительность INSERT в TokuDB vs. InnoDB [7]

Основное преимущество использования реляционных систем при работе с временными рядами – это, естественно, их доступность. Хостинг есть практически у любого провайдера, большое количество специалистов и, соответственно, меньше проблем с сопровождением.

С точки зрения обработки данных можно отметить расширения SQL, призванные облегчить работу с временными рядами. Например, аналитика для временных рядов от Vertica [8]. Здесь присутствуют два основных

момента: интерполяция для вычисления пропущенных значений групировка данных по времени. Вот типичный пример:

```
SELECT item, slice_time, ts_first_value(price, 'const') price FROM ts_test
WHERE price_time BETWEEN timestamp '2015-04-01 07:00' AND timestamp
'2015-04-01 07:30' TIMESERIES slice_time AS '1 minute' OVER (PARTITION
BY item ORDER BY price_time)  ORDER BY item, slice_time, price;
```

Такой запрос должен заполнить пропущенные данные в интервале 07:00 – 07:30 с шагом одна минута (slice_time as ‘1 minute’).

Другой похожий пример – это window functions в PostgreSQL [9]. Такие функции позволяют выполнять вычисления над набором строк в таблице, которые как-то относятся к текущей строке. Отчасти, это похоже на агрегатные функции. Но в отличие от обычных агрегатных функций, использование window function не группирует строки в одну. Это просто способ при вычислениях получить доступ сразу к нескольким строкам таблицы, а не только к текущей строке запроса. Типичный пример – скользящее среднее по трем строкам:

```
SELECT id_sensor, name_sensor, temperature,
        avg(temperature) OVER (ORDER BY id_sensor  ROWS BETWEEN 1
PRECEDING AND 1 FOLLOWING)
    FROM temperature_table;
```

Временные ряды в NoSQL системах

На концептуальном уровне, NoSQL как подход провозглашает отказ от единой модели данных (например, реляционной) и предлагает выбирать модели данных в зависимости от задачи. В этом плане, решения, предлагаемые для работы с временными рядами в NoSQL можно описать как,

в некотором роде, лучшие (на сегодняшний день, по крайней мере) решения по представлению временных рядов вне реляционной модели. Одним из таких часто используемых решений является OpenTSDB [10]. На рисунке 3 показана общая архитектура системы

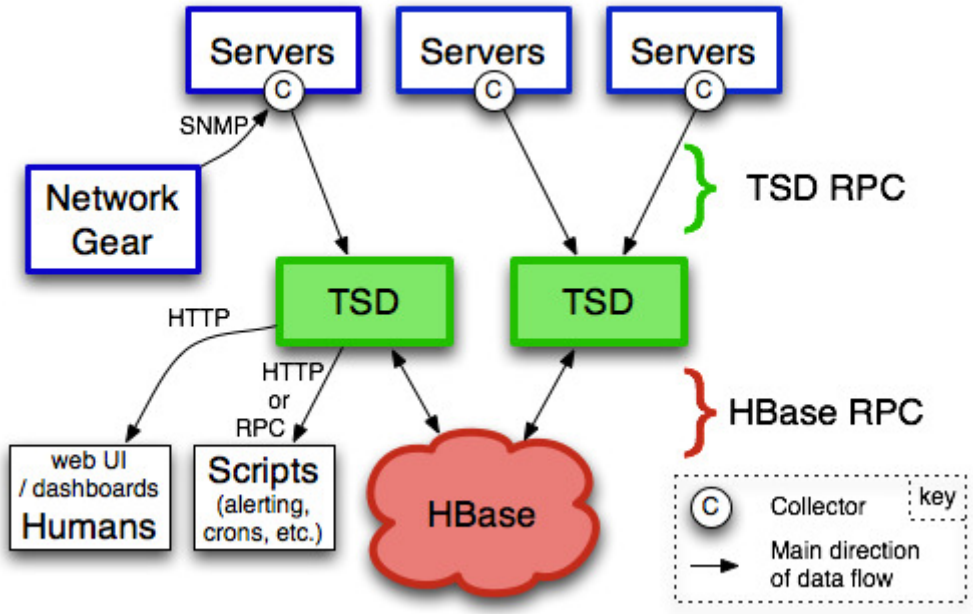

Рис. 3 Архитектура OpenTSDB [11]

OpenTSDB представляет собой набор так-называемых Time Series Daemon (TSD) и командных утилит. Демоны полностью независимы друг от друга. Каждый демон есть просто оболочка для доступа к HBase [12]. Демоны используют HBase для хранения данных, а также поддерживают открытые протоколы для доступа к данным. Иными словами, OpenTSDB – это схема (архитектура) использования HBase для хранения временных рядов. Эта схема включает в себя интерфейсные элементы (TSD) и модель представления данных в HBase. Независимость демонов нужна для горизонтальной масштабируемости при увеличении количества потоков данных. Если скорость поступления данных не слишком велика (хватает одного “демона”), то, как правило, и не будет необходимости в NoSQL решении.

Схема данных, примененная в OpenTSDB, также является, по сути, стандартом де-факто для преставления временных рядов в модели данных BigTable (это то, что является базисом HBase). Ее основная особенность – это возможность быстрой агрегации подобных временных рядов. В OpenTSDB элемент временного ряда (измерение) состоит из имени метрики (названия измерения), временной метки, значения и набора тегов (пар ключ – значение). Например (временной ряд, в котором собираются данные от датчиков температуры), имя метрики – *data.test*, ключ – *sensor_id*, значение ключа – некоторый идентификатор (адрес) сенсора. Основная таблица в HBase тогда выглядит так:

data.test, время, значение_температуры, sensor_id, адрес сенсора

Как обычно, в NoSQL системах отдельного уровня API не предусмотрено (по крайней мере, он не является обязательным) и описанная выше модель и определяет тот формат, в котором приложение будет записывать данные. Соответственно, все возможные приложения будут использовать именно такой формат. Никаких преобразований схем (как в реляционной модели) не требуется. Что, естественно, позволяет говорить об эффективной реализации операции добавления, которая, как было отмечено выше, является основной при работе с временными рядами.

В плане построения схемы описания данных, основная идея OpenTSDB состоит именно во введении тегов. Да, у метрик есть имя, но оно может быть одним и тем же у нескольких рядов. Уникальность измерения обеспечивается именно сочетание ключей (пар ключ-значение). Каждый временной ряд должен иметь, по крайней мере, один тег. Измерения для одного тега хранятся последовательно, что и обеспечивает быструю агрегацию данных. А множественные теги предназначены для реализации мультивариантных временных рядов.

Поскольку подобная модель представления временных рядов в NoSQL системах стала некоторым фактическим стандартом, то существуют решения и для других реализаций модели BigTable. Одним из наиболее известных (часто применяемых) примеров такой реализации служит Cassandra. Для временных рядов KairosDB [13] предложил свою реализацию OpenTSDB, используя Cassandra как хранилище данных.

Другой интересный момент в связке NoSQL и временные ряды состоит в растущей поддержке со стороны производителей измерительных устройств языка (системы) SenML [14]. SenML определяет модель данных для измерений и набор мета-данных для описания измерений. Единица представления в SenML – это объект с набором атрибутов. Объект содержат массив сущностей (измерений). Каждая сущность (измерение) имеет такие атрибуты как уникальный идентификатор, время и измеренное значение (Рисунок 4). SenML документ может быть представлен в виде JSON или XML текста:

```
<?xml version="1.0" encoding="UTF-8"?>
  <senml xmlns="urn:ietf:params:xml:ns:senml"
        bn="urn:dev:ow:10e2073a01080063" >
    <e n="voltage" t="0" v="120.1" u="V" />
    <e n="current" t="0" v="1.2" u="A" />
  </senml>
```

NoSQL решение для преставления временных рядов Geras DB [15] использует SenML как формат представления данных.

Следующий важный момент для NoSQL систем, работающих с временными рядами, состоит в поддержке MQTT. MQTT представляет собой популярный коммуникационный протокол для межмашинных коммуникаций (M2M) и Интернета Вещей (IoT) [16]. Протокол был специально разработан

для устройств с ограниченными вычислительными возможностями. MQTT используется как транспортный протокол для реализации моделей publish/subscribe. Сенсоры (измерительные устройства) публикуют свои данные, база данных в данном случае выступает как подписчик и должна читать опубликованные данные для их последующего хранения. Это соответствует упомянутой выше лямбда-архитектуре. Применительно к общей схеме работы с большими данными, в качестве шины сообщений может использоваться, например, Kafka [17], а в связи с временными рядами возникает потребность в поддержке именно шины MQTT.

Вне привязки к OpenTSDB, Cassandra [18] есть, пожалуй, лучший выбор для хранения временных рядов в NoSQL системах. Базовый принцип представления временных рядов в системах с хранением данных по столбцам достаточно очевиден. Мы можем создать группу столбцов с типом *TimeUUID* (это уникальный идентификатор и временная метка). Ключ для строки – имя для нашего измерения, имя столбца – это, по сути, время измерения, значение столбца – измеренное значение. В таком случае будет легко выбрать как индивидуальное значение какого-либо измерения, так агрегировать данные для заданного интервала. Важно еще, что упорядочивание данных (сортировка по времени, в данном случае) автоматически поддерживается самой системой Cassandra. Технически Cassandra позволяет хранить до 2 миллиардов столбцов в одной строке (два миллиарда отсчетов для временного ряда). На самом деле, данные поступающие с высокой частотой (а именно о таких системах и идет речь в NoSQL решениях) могут превысить это значение. Но в Cassandra этот вопрос можно решить, группируя данные в различных строках. Например, по дням: вместо одной строки для *sensor1*, хранить измерения по дням с ключами sensor1_20150402, sensor1_20150403 и так далее (суффикс соответствует дате).

Согласно классическому определению, модель использования больших данных описывается как 3V: Variety, Velocity and Volume (изменчивость

данных, скорость их поступления и объем). В случае временных рядов данные однородны, объем, в силу специфики обработки, сказывается только на архивном хранении, поскольку реально обрабатывается всегда только небольшая часть. Соответственно, скорость поступления данных является определяющим фактором в выборе между традиционными системами и NoSQL решениями. Из последнего класса, предпочтительным выбором, по нашему мнению, является Cassandra.

Заключение

В настоящей работе рассмотрены вопросы использования баз данных для хранения и обработки временных рядов. Целью являлось обоснование выбора системы хранения данных для задач Интернета вещей и межмашинного взаимодействия. В работе проанализированы решения, основанные на использовании традиционных реляционных баз данных, а также NoSQL решения. Из NoSQL систем предпочтительным выбором для хранения и обработки временных рядов является Cassandra.

Библиография

[1] Müller M. Dynamic time warping //Information retrieval for music and motion. – 2007. – С. 69-84.

[2] Fan W., Bifet A. Mining big data: current status, and forecast to the future //ACM SIGKDD Explorations Newsletter. – 2013. – Т. 14. – №. 2. – С. 1-5.

[3] Namiot D., Sneps-Sneppe M. On IoT Programming //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 10. – С. 25-28.

[4] Namiot D., Sneps-Sneppe M. On Micro-services Architecture //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 9. – С. 24-27.

[5] Wang J. et al. pLSM: A Highly Efficient LSM-Tree Index Supporting Real-Time Big Data Analysis //Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual. – IEEE, 2013. – С. 240-245.

[6] Bender, M. A.; Farach-Colton, M.; Fineman, J.; Fogel, Y.; Kuszmaul, B.; Nelson, J. (June 2007). "Cache-Oblivoius streaming B-trees". Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (CA: ACM Press): 81–92.

[7] TokuDB benchmark http://www.tokutek.com/2012/09/three-ways-that-fractal-tree-indexes-improve-ssd-for-mysql/.

[8] Bear, C., Lamb, A., & Tran, N. (2012, September). The vertica database: Sql rdbms for managing big data. In Proceedings of the 2012 workshop on Management of big data systems (pp. 37-38). ACM.

[9] Douglas K., Douglas S. PostgreSQL: a comprehensive guide to building, programming, and administering PostgreSQL databases. – SAMS publishing, 2003.

[10] Prasad S., Avinash S. B. Smart meter data analytics using OpenTSDB and Hadoop //Innovative Smart Grid Technologies-Asia (ISGT Asia), 2013 IEEE. – IEEE, 2013. – C. 1-6.

[11] How does OpenTSDB work? http://opentsdb.net/overview.html

[12] George L. HBase: the definitive guide. – " O'Reilly Media, Inc.", 2011.

[13] Leighton B. et al. A Best of Both Worlds Approach to Complex, Efficient, Time Series Data Delivery //Environmental Software Systems. Infrastructures, Services and Applications. – Springer International Publishing, 2015. – C. 371-379.

[14] Mosser S. et al. From Sensors to Visualization Dashboards: Need for Language Composition //Proceedings of the 2nd International Workshop on Globalization of Modeling Languages at MODELS. – 2013. – C. 6.

[15] Geras DB http://1248.io/geras.php

[16] Hunkeler U., Truong H. L., Stanford-Clark A. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks //Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on. – IEEE, 2008. – C. 791-798.

[17] Chardonnens T. et al. Big data analytics on high Velocity streams: A case study //Big Data, 2013 IEEE International Conference on. – IEEE, 2013. – С. 784-787.

[18] Lakshman A., Malik P. Cassandra: a decentralized structured storage system //ACM SIGOPS Operating Systems Review. – 2010. – Т. 44. – №. 2. – С. 35-40.