ОПТИМИЗАЦИЯ ХРАНЕНИЯ UUID

Д. И. Богомья
Кафедра информационных технологий автоматизированных систем, Белорусский государственный университет информатики и радиоэлектроники
Минск, Республика Беларусь
E-mail: dmitry.bogomya@gmail.com

Рассматриваются плюсы и минусы использования UUID в качестве глобально-уникального идентификатора. Предложен способ оптимизации хранения UUID в системах баз данных.

Введение

UUID (Universally Unique Identifier) – это стандарт идентификации, используемый в создании программного обеспечения. Основное назначение UUID — это позволить распределённым системам уникально идентифицировать информацию без центра координации. Таким образом, возможно создание UUID и использование его для идентификации чего-либо с приемлемым уровнем уверенности, что данный идентификатор непреднамеренно никогда не будет использован для чего-то ещё. Поэтому информация, помеченная с помощью UUID, может быть помещена позже в общую базу данных, без необходимости разрешения конфликта имен.

UUID представляет собой 16-байтное (128-битное) число, которое зачастую хранится в СУБД в текстовом виде, и используется в качестве первичного ключа. Такое применение дает нам уникальность первичных ключей для каждой таблицы, каждой базы данных и каждого сервера и позволяет легко проводить слияние записей из различных баз данных.

Существует несколько способов генерации UUID. В этом докладе будет рассмотрен способ генерации уникальных идентификаторов на основе комбинации метки времени и MAC-адреса компьютера, на котором генерируется UUID, и будет описана стратегия оптимизации хранения UUID.

I. Проблемы при использовании UUID

Использование UUID как первичного ключа вызывает ряд проблем.

- Длина стандартного UUID 128 бит (зачастую идентификаторы хранятся в виде строки шестнадцатеричных цифр, что занимает 32 байта)
- Отсутствие естественной сортировки ключей;
- Случайные вставки и разброс данных.

Именно тот факт, что вставки идут в случайные места в дереве индекса, требуется совершать множество операций чтения/записи в случае, если дерево индекса не будет вписываться в памяти. Следует также отметить, что зачастую в системах управления базами данных вторичные ключи содержат внутри себя значения первичного ключа. Таким образом, наличие UUID как первичного ключа увеличивает размер вторичных индексов.

Несмотря на проблемы с UUID, использование его в базах данных— отличный выбор, потому что это единственные способ получить глобально-уникальный идентификатор.

II. Оптимизация хранения UUID

Хранение UUID в MySQ организовано в виде 128-разрядного числа, представленного в utf8 строке из пяти шестнадцатеричных чисел. В MySQL используется UUID версии 1.

- Первые три числа генерируются на основе метки времени;
- Четвертое число сохраняет временную уникальность в том случае, если значение временной метки теряет свою последовательность (например, из-за перехода на летнее время);
- Пятое число является IEEE 802 номером узла, что обеспечивает пространственную уникальность. Если узел не доступен, подставляется случайное число.

В докладе рассмотрен пример: пусть шестнадцатеричное значение метки времени равно «1e5651d15f632c6». Это значит, что первые 3 числа нашего UUID будут иметь вид в «15f632c6-651d-11e5». Первая цифра в третьем числе дополнительно указывает нам версию UUID, В нашем случае версия равна 1. Если мы будем генерировать UUID с одного и того же сервера, четвертая и пятые цифры как правило будут постоянными. Таким образом наш UUID может иметь следующий вид: «15f632c6-651d-11e5-9d70-feff819cdc9f».

Предлагаемой стратегией оптимизации хранения UUID является изменение порядка цифр в UUID с целью сделать генерируемые значения последовательными. Поскольку первые три цифры основаны на временной метке, таким образом, если поставить их в противоположном порядке, значения генерируемых UUID будут монотонно расти. Это значительно ускорит вставки вставку данных. Следует хранить UUID в виде бинарных строк, чтобы иметь возможность вести числовое сравнение. Символы тире («-») в UUID не имеют смысла, поэтому можно удалить их. После преобразование мы получим число следу-

ющего вида «11e5651d15f632c69d70feff819cdc9f». Чтобы продемонстрировать разницу между изначальным и измененным UUID, было создано 3 таблицы, в которые. В первой – уникальный ключ был числовым автоинкрементным значением

III. Сравнение производительности

Демонстрация изменений в производительности между изначальным и измененным UUID, продемонстрирована путем создания 3 таблиц, в которые велась пакетная вставка данных (по 5000 записей). В первой таблице первичный ключ является числовым автоинкрементным полем, во второй – полем, в которое записываются оригинальные значения UUID, в третьей – поле, в которое записываются значения UUID, оптимизированные предложенным способом. По результатом тестов был построен график зависимости времени вставки данных от количества пакетных вставок (см. рис. 1).

Для таблицы с UUID в качестве первичного ключа время, необходимое для вставки строк растет почти линейно. В то время как для других таблиц, время, необходимое для вставки данных практически постоянно.

Заключение

Использование UUID в качестве первичного ключа – отличный способ получить идентификатор, который будет уникален для всей системы целиком. Предложенный алгоритм оптимизации позволяет сократить издержки при использовании UUID в качестве первичного ключа.

1. Ткаченко, В. MySQL. Оптимизация производительности / В. Ткаченко – М. : Символ . – 2010. – 823 с.
2. Талманн, Л. Обеспечение высокой доступности систем на основе MySQL / Л. Талманн, Ч. Белл // БХВ-Петербург. – 2012. – 624 с.
3. Myths, GUID vs Autoincrement [Электронный ресурс] / В. Aker. – Washington 2007. – Режим доступа: http://krow.livejournal.com/497839.html. – Дата доступа: 25.09.2015.

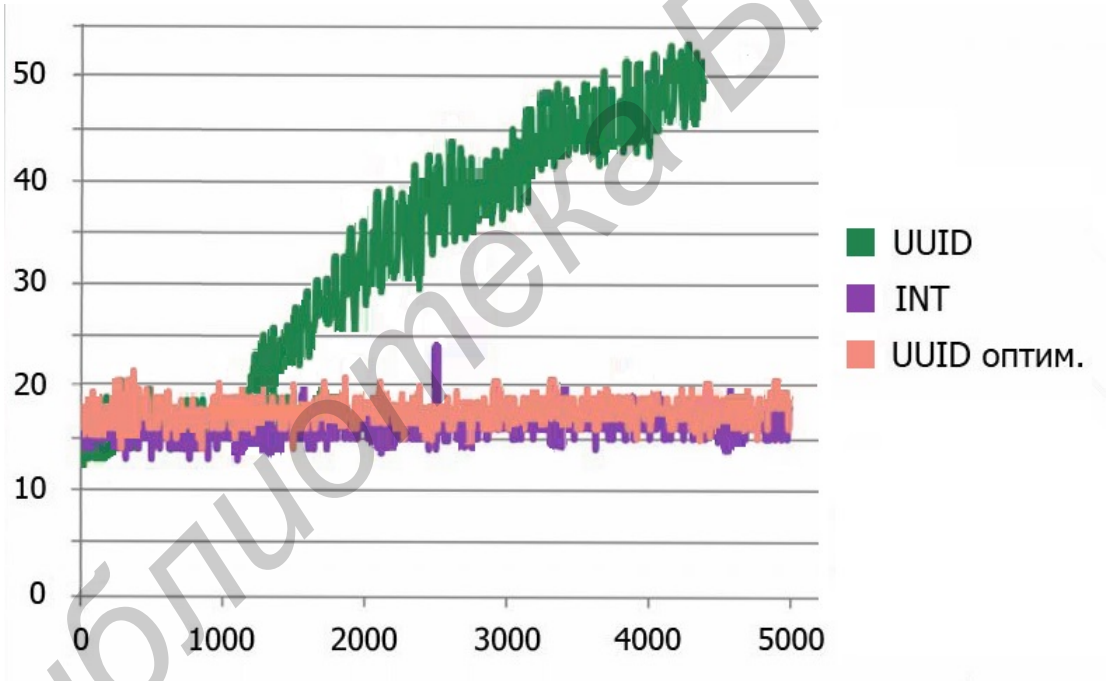

Рис. 1 – Зависимость времени вставки данных от количества пакетных ставок