Каждый объект на логическом уровне хранится в конкретной плейсмент-группе. На физическом же уровне он лежит в нужном количестве копий на разных OSD (физический диск, RAID-массив или iSCSI-устройство), которые в эту плейсмент-группу включены.

Монитор — это служба, выполняющая роль координатора, с которого начинается кластер. Как только появляется хотя бы один рабочий монитор, появляется Ceph-кластер. Монитор хранит информацию о здоровье и состоянии кластера, обмениваясь различными картами с другими мониторами. Клиенты обращаются к мониторам, чтобы узнать, на какие OSD писать/читать данные. При разворачивании нового хранилища первым делом создается монитор (или несколько). Кластер может прожить на одном мониторе, но рекомендуется делать три или пять мониторов во избежание падения всей системы по причине падения единственного монитора. Необходимо, чтобы количество мониторов было нечетным [1].

OSD (Object Storage Device) — это юнит хранилища, который хранит сами данные и обрабатывает запросы клиентов, обмениваясь данными с другими OSD. Обычно это диск. И обычно за каждый OSD отвечает отдельный OSD-демон, который может запускаться на любой машине, на которой установлен этот диск. Это второе, что нужно добавлять в кластер при первичной настройке.

В основе механизма децентрализации и распределения лежит CRUSH-алгоритм (Controlled Replicated Under Scalable Hashing), играющий важную роль в архитектуре системы. Этот алгоритм позволяет однозначно определить местоположение объекта на основе хеша имени объекта и определенной карты, которая формируется исходя из физической и логической структур кластера (датацентры, залы, ряды, стойки, узлы, диски). Карта не включает в себя информацию о местонахождении данных [1]. Путь к данным каждый клиент определяет сам, с помощью CRUSH-алгоритма и актуальной карты, которую он предварительно спрашивает у монитора. При добавлении диска или падении сервера карта обновляется.

Ceph предусматривает несколько способов увеличения производительности кластера методами кеширования: Primary-Affinity, вынос журналов на SSD, кеш-тиринг.

Клиент может смонтировать файловую систему CephFS, если у него Linux с версией ядра 2.6.34 или новее. Если версия ядра старше, то можно смонтировать ее через FUSE (Filesystem in User Space). Для того чтобы клиенты могли подключать Ceph как файловую систему, необходимо в кластере поднять хотя бы один сервер метаданных (MDS) [3].

Удаление данных непосредственно с диска — это довольно ресурсоёмкая задача, в продвинутых системах удаление делается отложено или не делается вообще. Ceph — тоже продвинутая система, и в случае с RGW при удалении s3-объекта соответствующие RADOS-объекты не удаляются с диска сразу [3]. RGW помечает s3-объекты как удаленные, а отдельный gc-поток занимается удалением объектов непосредственно из RADOS-пулов и, соответственно, с дисков отложено.

Заключение. Так как Ceph лёгок в настройке и обеспечивает постоянный доступ к информации, часто используется хостинг-провайдерами, в таком случае клиент не беспокоится о проблемах с недоступностью и потерей данных. Для клиента важным является то, что данные не хранятся в одной точке, копии находятся на разных стойках (возможно даже в разных датацентрах).

Список цитируемых источников

1. *Д'Атри, Э.* Изучаем Ceph. / Э. Д'Атри. — 2-е изд. — М. : Модуль-Проекты, 2017.
2. *Сингх, К.* Книга рецептов Ceph / К. Сингх. — М. : Модуль-Проекты, 2016.
3. *Фиск, Н.* Полное руководство Ceph / Н. Фиск. — Packt Publishing Ltd, 2017.

УДК 004.657

И. В. Яковюк

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники», Минск

МЕТОДЫ ОПТИМИЗАЦИИ ЗАПРОСОВ К БАЗЕ ДАННЫХ И ЗАЩИТА ОТ SQL-ИНЪЕКЦИЙ

Введение. Частой причиной медленной работы приложения, интернет-ресурса кроме ошибок, допущенных при настройке сервера, является наличие неоптимизированных, медленных запросов к базе данных. При долгом выполнении SQL-запроса конечный клиент долго ожидает ответа от сервера, к тому же в этот момент возрастает нагрузка на самом сервере. Для предотвращения указанных и многих других проблем необходимо провести оптимизацию.

Основная часть. SQL server management studio позволяет получать дополнительную информацию о выполнении запроса: план запроса получается функциями “Display Estimated Execution Plan” (оценочный план) и “Include Actual Execution Plan” (фактический план). Отличаются они тем, что оценочный план строится без выполнения запроса. Соответственно, информация о количестве обработанных строк в нем будет только оценочная. В фактическом плане будут как оценочные данные, так и фактические. Сильные расхождения этих величин говорят о неактуальности статистики.

Также можно получать замеры затрат процессора и дисковых операций сервера. Для этого необходимо включить SET-опции либо в диалоге через меню “Query” / “Query options...” [1].

В результате выполнения будут данные по затратам времени на компиляцию и выполнение, а также количество дисковых операций.

Здесь стоит обратить внимание на время компиляции и текст «логических чтений N, физических чтений M». При втором и последующих выполнениях одного и того же запроса физические чтения могут уменьшаться, а повторная компиляция может не потребоваться. Из-за этого часто возникает ситуация, что второй и последующие разы запрос выполняется быстрее, чем первый. Причина — в кешировании данных и скомпилированных планов запросов [1].

1. Затраты процессора можно смотреть, используя SET STATISTICS TIME ON.
2. Дисковые операции: SET STATISTICS IO ON. «Логическое чтение» — это операция чтения, завершившаяся в кеше диска без физического обращения к дисковой системе. «Физическое чтение» требует значительно больше времени.

Объем сетевого трафика оценивается с помощью “Include Client Statistics”.

Детально алгоритм выполнения запроса анализируется по «плану выполнения» с помощью “Include Actual Execution Plan” и “Include Live Query Statistics”.

Анализ нагрузки от приложения.

Для анализа нагрузки, создаваемой приложением, необходимо воспользоваться SQL Server Profiler.

В первую очередь происходит запуск SQL Server Profiler и подключение к серверу. Затем необходимо выбрать журналируемые события. Наиболее простой вариант — запустить профилирование со стандартным шаблоном трассировки. На закладке “General” в поле “Use the template” выбрать “Standard (default)” и нажать “Run” [1].

Другой вариант — к выбранному шаблону добавить (или убавить) фильтры или события. Данные опции на второй закладке диалога. Чтобы увидеть полный набор возможных событий и колонок для выбора, нужно отметить пункты “Show All Events” и “Show All Columns”.

Из событий потребуются: Stored Procedures \ RPC:Completed; TSQL \ SQL:BatchCompleted.

Эти события фиксируют все внешние sql-вызовы к серверу. Они возникают, как видно из названия (Completed), после окончания обработки запроса. Имеются аналогичные события фиксирующие старт sql-вызова: Stored Procedures \ RPC:Starting; TSQL \ SQL:BatchStarting [1].

Очевидно, что такая информация доступна только по окончании выполнения. Соответственно, столбцы с данными по CPU, Reads, Writes в событиях *Starting будут пустыми.

Множества трудностей можно избежать, если не допускать ошибки на этапе проектирования и разработки приложения.

Как отмечается в исследованиях, следует помнить о специальном значении NULL. NULL ещё называется UNKNOWN. Причина в том, что при получении данных и связывании переменных JDBC отражает SQL NULL в Java null. Это может привести к тому, что NULL = NULL (SQL) будет вести себя так же, как и null == null (JAVA) [2].

Часто разработчики загружают SQL-данные в память, трансформируют их в подходящую коллекцию и выполняют нужные вычисления на этих коллекциях с многословными циклическими структурами. Важным является использование OLAP-функций, которые подходят для этого лучше и являются более простыми в написании [2].

Большинство баз данных поддерживают средства для постраничной разбивки через LIMIT… OFFSET, TOP… START AT, OFFSET… FETCH операторов. В отсутствии поддержки этих операторов всё ещё есть возможность наличия ROWNUM (Oracle) или ROW_NUMBER() OVER() фильтрации (DB2, SQL Server 2008 и др.), которые намного быстрее разбивки в памяти. Использование JDBC для постраничной разбивки большой выборки является более грамотным решением.

В исследованиях по безопасности отмечается, что для защиты от SQL-инъекций следует придерживаться следующий правил при разработке приложения: подставлять данные в запрос только через плейсхолдеры; идентификаторы и ключевые слова подставлять только из белого списка, прописанного в коде [3].

Любые данные должны попадать в запрос не напрямую, а через представителя, подстановочное выражение. В общем виде запрос будет иметь вид: SELECT * FROM table WHERE id > ? LIMIT ?

А данные добавляются и обрабатываются отдельно.

Разработчики сталкиваются с необходимостью подставлять в запрос не только данные, но и другие элементы: идентификаторы (имена полей и таблиц) и даже элементы синтаксиса, ключевые слова. Даже незначительные, как DESC или AND, но требования к безопасности таких подстановок всё равно должны быть не менее строгими [3].

Заключение. Приведём пример: имеется база элементов, которая выводится пользователю в виде HTML-таблицы. Пользователь может сортировать эту таблицу по одному из полей, в любом направлении.

Со стороны пользователя к базе приходит имя колонки и направление сортировки. Подставление их в запрос напрямую приведёт к инъекции. Рекомендуемое решение — белый список.

Суть метода заключается в том, что все возможные варианты выбора должны быть жёстко прописаны в коде, и в запрос должны попадать только они, на основании пользовательского ввода.

Список цитируемых источников

1. *Бен-Ган, И.* Microsoft SQL Server 2012. Основы T-SQL / И. Бен-Ган. — СПб. : БХВ-Петербург, 2015. — 400 с.
2. *Шварц, Б.* MySQL по максимуму / Б. Шварц, П. Зайцев, В. Ткаченко. — СПб. : Питер, 2018. — 864 с.
3. *Тахагхогхи, С.* Руководство по MySQL / С. Тахагхогхи, Х. Е. Вильямс. — СПб. : Рус. Ред., 2007. — 544 с.