Възможности за използване на системи от тип NoSQL за увеличаване на производителността на информационните системи

Павел Петров

Some possibilities for using NoSQL systems to increase the performance of information systems

Pavel Petrov

Abstract

The study focuses on possibilities of NoSQL systems to be used as cache layer between an application and RDBMS SQL systems. Series of benchmark tests has proceeded with aim to determine when there will be advantage of using such approach to save hardware resources. The term "relative performance" has been defined and this indicator is used to determine relative performance between popular SQL и NoSQL systems such as MySQL and Redis

Key words: NoSQL, MySQL, Redis, performance

При големи обеми данни скоростта при обработване на заявките понякога се явява от висока степен на важност за една информационна система и минимизирането на времето за отговор може да се реализира посредством кеш. Резултатите от заявките се съхраняват в хранилище с многократно по-малко време за отговор от тип NoSQL, в който да се намира определено подмножество от всички данни в базата от данни. Ефектът ще е по-голям там, където се кешират данни при високо натоварени системи и там където извличането на данните изисква големи изчислителни ресурси.

Изследване на компанията Market Research Media прогнозира среден ръст на пазара на NoSQL системи от 21% всяка година в периода 2013-2018 г.[1], което ще е резултат от инвестициите в този сектор и повишеното използване на този вид системи. Тенденцията е при нарастване на обемите обработвани данни, да се търсят по-оптимални решения.

Операциите, които могат да се извършват с хеш-таблиците, са сравнително елементарни:

- set - задаване на стойност на ключ;
- get - връщане на стойност по ключ;
- delete - изтриване на стойност по ключ.

При някой системи (най-често кеширащи в оперативната памет), се задава и време на живот на ключ, след изтичането на което ключа и стойността автоматично се изтриват.

Използването на кеша става по следния начин: заявките подобни на "SELECT name FROM users WHERE id = 111;" са ключ в хеш-таблицата, а резултатът от изпълнението на заявката - стойност, съответстваща на ключа. Ако ключът съществува, се използва кеширания резултат. Ако ключа не съществува, се изпраща заявката до СУРБД и тогава заявката и резултата се добавят в хеш-таблицата. Резултатът обикновено се записва като низ, което налага данните да се сериализират по подходящ начин, за да могат да се извличат отделни колони и редове.

При този подход е добре да се кешират резултантни таблици с повече записи, вместо с един или няколко, тъй като комбинираното натоварване при извличане на множество записи поединично е по-голямо от груповото им извличане на един път. Например заявката "SELECT name FROM users WHERE id = 111;" може да се преобразува в "SELECT name FROM users

[1] Market Research Media, NoSQL Market Forecast 2013-2018, <http://www.marketresearchmedia.com/?p=568>, (05.10.2013)

WHERE id >= 100 and id <= 200;" с последващо извличане на нужния запис по програмен път от кеша. По този начин се увеличава вероятността търсените данни да се намират в кеша.

Кеширането може да се извършва както в оперативната памет, така и на твърдия диск. Неудобството при кеширане в паметта е, че липсва постоянно съхраняване на данните в кеша, при което натоварването в първоначалния момент при запълване на кеша се увеличава[2]. Съхраняването на данните на кеш във файл спрямо паметта осигурява сравнително постоянна производителност през големи интервали от време и възможност за съхраняване на по-големи по обем данни. Недостатък е по-ниската скорост на работа, свързана с по-ниската скорост на обмен на данни от периферните устройства спрямо оперативната памет.

Трябва да се има в предвид, че кеширането на резултатите от някои заявки може да обезсмисли използването на кеш, което най-често се случва когато данните по-често се обновяват и по-рядко се четат. В този случай те не трябва да се кешират. Например, ако заявките за извличане се изпълняват през определен интервал от време, през който имаме редица обновявания. Видно е, че в този случай вероятността за използване на данните от кеша е много ниска. Друг вариант е, когато заявката извлича данни от множество таблици, или от една, но с много голям брой записи и вероятността някой от записите да бъде променен е висока.

Добре е в програмния код предварително да се предвиди използването на кеш, отколкото в последствие да се правят промени в него.

Някой изследователи публикуват удивителни резултати, като например в едно изследване на създателя на системата за съхранение на данни на Facebook - Casandra и на Amazon - Dynamo, се твърди, че скоростта при запис в NoSQL Casandra е 2500 пъти по-голяма спрямо тази в MySQL, а при четене - около 23 пъти.[3]

Според нас голяма част от тестовете, които са опитват точно да определят разликите в производителността между СУРБД и NoSQL-системите не са коректни, тъй като могат да разкрият само общите тенденции, но не и да се определи точно колко едните са по-производителни от другите. Причините за това са, че проблемните области, в които се използват системите са различни, SQL-заявките, които се отправят биха могли да са както сравнително по-елементарни, така и с по-сложен синтаксис, включващ работа с няколко таблици и използващи специфичните възможности на релационната алгебра. За сравнение заявките към NoSQL системите са по-опростени и в най-елементарен вид се свеждат до get/set операции.

В някои изследвания се предлага методологията за тестване на производителността да включва точно определени параметри, които трудно биха могли да се срещнат точно в този вид в практиката. Например някои автори предлагат при тестовете да се ползват таблици, чиито записи са с дължина кратна на 32 KB, като се състоят от 3 32-битови полета, 3 полета от числа с плаваща запетая, 3 текстови полета от 100 символа, 1 поле от тип малък BLOB (Binary Large OBject). Като допълнителен параметър се въвежда съотношението брой операции четене/запис, който се предлага да е 500:1. Последното е предложено на база информация, че в YouTube съотношението четене/запис е около 1389:1[4]. Очевидно е, че при тези тестове се създава една

[2] Този проблем, наречен "студен кеш" (Cold Cache) (първоначално празен кеш) е описан подробно във връзка с използването на memcached клъстер в сайта SixApart. Първоначалното запълване на кеше наподобява атака от вида "Отказ на обслужване" за сървърите. За избягване на тази ситуация предварително кеша постепенно се "подгрява" предварително с заявки (клонирани или симулирани) и след това се включва за нормална работа.

[3] Avinash Lakshman, Prashant Malik Cassandra. Structured Storage System over a P2P Network, <http://odbms.org/download/cassandra.pdf>, (05.10.2013)

[4] Ion LUNGU, Bogdan George TUDORICA, The Development of a Benchmark Tool for NoSQL Databases, Database Systems Journal vol IV, no. 2/2013, <http://www.dbjournal.ro/archive/12/12_2.pdf>, (05.10.2013)

специфична среда, която се опитва да се доближи до практиката, но тъй като последната е прекалено разнообразна, то това не е възможно.

Според нас е по-правилно да се определят диапазони от резултати, на базата на тестове при гранични условия - най-лош и най-добър случай, при което очакванията за разлика в производителността при реална система ще попадат в така определения диапазон. Освен това резултатите трябва да се представят в относителни, а не в абсолютни стойности във вид на "брой операции за единица време". Предлагаме този показател да се нарича "относителна производителност", при което се съпоставят резултатите между две системи, получени при тестване при максимално близки хардуерни, софтуерни и други условия. В идеалния случай това означава изпълнение на тестовете по едно и също време, на един и същи хардуер, с общ език за програмиране и тип на извършвани операции. В зависимост от типа и сложността на операциите е възможно да се получат различни резултати, тъй като едни системи са по-добре оптимизирани да изпълняват определени дейности, докато други системи - други дейности. В резултат се получават стойности, които варират в широки граници, но в определен диапазон. Според нас границите на този диапазон могат да бъдат определени, като това ще улесни определянето в кои случаи има смисъл да се прилага кеширане.

В нашето изследване сме се опитали да определим долната граница на диапазона за относителна производителност при използване на СУРБД MySQL, съпоставено с NoSQL Redis. Причината да изберем тези две система са данните от изследвания за популярност. Така например според авторитетно изследване на австрийската консултантска фирма solidIT, поддържаща сайта http://db-engines.com, към м. Септември 2013 г, най-популярната релационна система за управление на бази от данни в отворен код е MySQL[5] (на първо място е Oracle, но тя не е с отворен код), а сред NoSQL системите от тип ключ-стойност с отворен код, най-популярна е Redis[6].

Разработили сме модул за тестване на бързодействието на MySQL и Redis на базата на Java, с цел приложението да може да се използва и на други платформи. Отчетени са някой особености при създаването на модули за тестване на Java[7], като е направен опит да се минимизират проблемите, свързани със зареждането на клас-файловете, действието на Just-In-Time компилатора, автоматичното освобождаване на ресурси и прочие.

Използваната Java виртуална машина е Java SE Runtime Environment (build 1.7.0_40-b43), Java HotSpot Client VM (build 24.0-b56, mixed mode, sharing), използвания компилатор - JDK SE 1.7.0_40. СУРБД MySQL е версия MySQL Server 5.6.11[8]. Като драйвер за връзка с MySQL е използван "JDBC Driver for MySQL 5.1.26"[9]. За NoSQL Redis сме използвали компилирана версия - Redis-2.4.5[10] с непроменяни настройки по подразбиране. За драйвер-клиент е използван Jedis 2.1.0[11]. Тестовете са проведени на ОС Windows 7 Professional, x86; процесор Intel i5-2430M@2.40GHz; RAM 4GB.

Програмният код за добавяне на записи и за четене на записи в MySQL е следния:

```
static long insertInMysql(int count) throws SQLException {
   Connection dbCon =
DriverManager.getConnection("jdbc:mysql://localhost:3306/benchmark",
"benchmark", "123");
```

[5] DB-Engines Ranking of Relational DBMS, <http://db-engines.com/en/ranking/relational+dbms>, (05.10.2013)
[6] DB-Engines Ranking of Key-value Stores, <http://db-engines.com/en/ranking/key-value+store>, (05.10.2013)
[7] Brent Boyer, Robust Java benchmarking, Part 1: Issues, 2008, <http://www.ibm.com/developerworks/java/library/j-benchmark1/j-benchmark1-pdf.pdf>, (05.10.2013)
[8] MySQL Community Server 5.6.14, <http://dev.mysql.com/downloads/mysql/>, (05.10.2013)
[9] Connector/J 5.1.26, <http://dev.mysql.com/downloads/connector/j/>, (05.10.2013)
[10] Redis-2.4.5 - Windows binaries (win32 and x64), <https://github.com/dmajkic/redis/downloads>, (05.10.2013)
[11] Jedis, <http://code.google.com/p/jedis/>, (05.10.2013)

```
   PreparedStatement  stmt;

   stmt = dbCon.prepareStatement("TRUNCATE TABLE test");
   stmt.executeUpdate();

   long start = System.currentTimeMillis();
   for(int i=1; i<=count; i++) {
      stmt = dbCon.prepareStatement("INSERT INTO test VALUES (" + i
+ ", '" + i + TXT + "')");
      stmt.executeUpdate();
      stmt.close();
   }
   long end = System.currentTimeMillis();

   dbCon.close();

   return end-start;
}

static long readFromMysql(int count) throws SQLException {
   Connection                          dbCon                          =
DriverManager.getConnection("jdbc:mysql://localhost:3306/benchmark",
"benchmark", "123");
   PreparedStatement  stmt;
   ResultSet rs = null;
   int id;

   long start = System.currentTimeMillis();
   for(int i=0; i<count; i++) {
      id = 1+i%200*i%200;
      stmt = dbCon.prepareStatement("SELECT data FROM test WHERE id
= " + id + " LIMIT 1");
      rs = stmt.executeQuery();
      rs.close();
      stmt.close();
   }
   long end = System.currentTimeMillis();

   dbCon.close();

   return end-start;
}
```

Използваната таблица е създадена по подобие на DBM базите от данни с две полета - целочислово - с първичен ключ и текстово от тип varchar. По този начин се доближаваме възможно най-близко до NoSQL системите от тип ключ/стойност и имаме възможност да сравним чистата производителност при достъп до данните, без да се използва сложна релационна алгебра.

При четене на данните се използва опростен алгоритъм за генериране на псевдослучайни числа с цел заявките да са за записи, които не са последователно разположени, по подобие на реално работещите системи. Не сме правили опити да елиминираме кеширането

от страна на процесора, на входно-изходните операции от страна на операционната система или от страна на контролера на твърдия диск.

Програмният код за добавяне и четене на данни в Redis е следния:

```
static long insertInRedis(int count) {
   Jedis jedis = new Jedis("localhost");
   jedis.connect();

   jedis.flushDB();

   long start = System.currentTimeMillis();
   for(int i=0; i<count; i++) {
      jedis.set("" + i, i + TXT);
   }
   long end = System.currentTimeMillis();

   return end-start;
}

static long readFromRedis(int count) {
   Jedis jedis = new Jedis("localhost");
   jedis.connect();
   String v = "";
   int id;

   long start = System.currentTimeMillis();
   for(int i=0; i<count; i++) {
      id = 1+i%200*i%200;
      v = jedis.get(""+id);
   }
   long end = System.currentTimeMillis();

   return end-start;
}
```

Както се вижда, чрез предаване на аргументи на функциите се задава броя на записаните записи/стойности и броя на прочетените записи/стойности (при всички тестове сме ползвали 10 000 четения). Останалата част от кода не го публикуваме, тъй като представлява импортиране на пакети, декларация на константи и извиквания на горепосочените функции, след изчакване от 10 секунди между отделните етапи на работа, за да може операционната система евентуално да завърши някои операции.

Тестовете са проведени по следната схема: по дадени настройки за брой записи/стойности приложението се стартира три пъти с цел "подгряване" на оперативната памет и периферните устройства. Всеки тест се стартира 5 пъти, след което най-малките и най-големите стойности се премахват и от останалите се намира средно аритметично. Броя записи/стойности нараства експоненциално от 1 до 1 000 000. Резултатите са показани в две таблици. В табл.1 са показани резултатите при запис, а в табл. 2 - резултатите при четене на 10 000 записа от MySQL или стойности от Redis. Флуктуациите в числата са в нормални граници при подобен род тестове.

Таблица 1

Относителна производителност на MySQL и Redis при запис

Брой записи/ стойности	MySQL		Redis		Относителна производителност при запис MySQL:Redis
	Време за изпълнение	Брой записи в секунда	Време за изпълнение	Брой записи в секунда	
1	0 ms	-	0 ms	-	-
10	31 ms	323	0 ms	-	-
100	156 ms	641	0 ms	-	-
1 000	1,3 s	745	47 ms	21 277	1:29
10 000	13 s	763	0,4 s	24 631	1:32
100 000	140 s	713	4 s	24 468	1:34
1 000 000	26,2 m	636	39 s	25 589	1:40

Таблица 2

Относителна производителност на MySQL и Redis при четене на 10 000 записа/стойности

Брой записи/ стойности	MySQL		Redis		Относителна производителност при четене MySQL:Redis
	Време за изпълнение	Брой четения в секунда	Време за изпълнение	Брой четения в секунда	
1	1,2 s	8 224	0,4 s	22 936	1:3
10	1,2 s	8 326	0,4 s	23 753	1:3
100	1,2 s	8 326	0,4 s	22 883	1:3
1 000	1,2 s	8 439	0,4 s	22 883	1:3
10 000	1,2 s	8 117	0,4 s	22 883	1:3
100 000	1,2 s	8 479	0,4 s	23 753	1:3
1 000 000	1,2 s	8 217	0,4 s	22 935	1:3

От направените изследвания можем да направим извода, че:

1. MySQL е десетки пъти по-бавен при запис на данни, спрямо Redis, като при увеличаване на броя записи в таблицата, тази тенденция се засилва. Долният диапазон за относителна производителност при запис MySQL:Redis е около 1:30 - 1:40, като при усложняване на структурата на добавяните записи (брой колони, тип данни, индексни полета и т.н.) този показател ще се влошава за MySQL, т.е. ще е над 1:40. Това се дължи както на сигурния начин, по който се изпълняват SQL-заявките, така и на богатите възможности на релационния модел.

2. Долният диапазон за относителна производителност при четене MySQL:Redis е около 1:3. И при двете системи се забелязва, че обема на съхраняваните данни почти не оказва влияние върху времето за търсене на нужния запис или ключ, и е постоянно. При увеличаване на сложността на заявките чрез използване на пълните възможности на SQL, този показател отново ще се влошава за MySQL, при което съотношението ще бъде над 1:3.

3. При положение, че записването на данни в Redis отнема незначително време спрямо подобна операция в MySQL, добавянето на кеширащ слой използващ NoSQL в едно приложение е обосновано при определена степен на вероятност данните да се намират в кеша и да не се налага изпращане на заявка до СУРБД. В този конкретен случай полза ще има при съотношение 1 запис, последван от поне 2,5 четения, т.е. при вероятност за намиране на данни

в кеша над 60%[12]. Приложими са следните разчети на база емпиричните стойности от табл.1 и табл. 2:

а) без кеширане в MySQL: 1 записване, последвано от "N" четения, повтарящо се 10000 пъти:

13 + 1,2xN

, т.е. при 1 четене (N=1) - времето за изпълнение ще е 14,2 s, при 2 четения (N=2) - 15,4 s, при 3 - 16,6 s, при 4 - 17,8 s, и т.н.

б) с кеширане чрез Redis, схемата е следната: 1 записване в MySQL, 1 изтриване в Redis. Следва алтернативно: 1 четене от Redis, 1 четене от MySQL, 1 записване в Redis, а при следващи четения - само 1 четене от Redis, като при "N"-четения първата алтернатива се изпълнява веднъж, а втората алтернатива се изпълнява N-1 пъти:

(13 + 0,4) + (0,4 + 1,2 + 0,4) + 0,4x(N-1)

, т.е. при 1 четене (N=1) - времето за изпълнение ще е 15,4 s, при 2 четения - 15,8 s, при 3 - 16,2 s, при 4 - 16,6 s, и т.н.

Вижда се, че при над 3 четения включително (точната стойност, както казахме по-горе е 2,5 четения) сумарното време за изпълнение при наличие на кеш започва да намалява спрямо случая без кеш.

Използвана литература

1. Avinash Lakshman, Prashant Malik Cassandra. Structured Storage System over a P2P; Network, <http://odbms.org/download/cassandra.pdf>, (05.10.2013);
2. Brent Boyer, Robust Java benchmarking, Part 1: Issues, 2008, <http://www.ibm.com/developerworks/java/library/j-benchmark1/j-benchmark1-pdf.pdf>, (05.10.2013);
3. DB-Engines Ranking of Key-value Stores, <http://db-engines.com/en/ranking/key-value+store>, (05.10.2013);
4. DB-Engines Ranking of Relational DBMS, <http://db-engines.com/en/ranking/relational+dbms>, (05.10.2013);
5. Ion LUNGU, Bogdan George TUDORICA, The Development of a Benchmark Tool for NoSQL Databases, Database Systems Journal vol IV, no. 2/2013, <http://www.dbjournal.ro/archive/12/12_2.pdf>, (05.10.2013);
6. Market Research Media, NoSQL Market Forecast 2013-2018, <http://www.marketresearchmedia.com/?p=568>, (05.10.2013).

За контакти:
Доц. д-р Павел Петров
Икономически университет – Варна
petrov@ue-varna.bg

[12] Получено е след приравняване на двата израза дадени по-долу и решаване на уравнението.