Search...
Go

Java
JAXB
JSF
JAX-WS

Tools
GlassFish
Maven
Ant

Databases

C#

C++

Web
Joomla

ant apache bash boost c sharp cpp css database glassfish h2database hotspot idea java javascript jax-ws jaxb joomla jquery jsf junit maven mongodb netbeans nosql oracle rich-faces rman solaris sql sqlplus tools web wsdl xml zfs

Blog Databases H2 performance tuning

H2 performance tuning

18 November 2012

H2 database engine is fast enough. But in my case its performance was not as good as I wished. This article is about my experience of speed improvements of H2 (version **1.3.169**).

In an application where H2 was used, the database size was small (on the drive it takes about 100 MB), but it received a quite large number of concurrent requests (about 10-15 queries per second).

Of course, first I profiled the application and the database system (next I will call it just database for short) - H2 has a good tool for this.

The results showed that 50% of time database spends on executing a quite complex SELECT-query, which contains two subqueries, where the first one joins 5 tables, and another - 7 tables. So I test ongoing optimizations, mostly on this query.

Indexes

This is the first thing advised to make by the H2 documentation.

To Learn exactly which indexes are missing, you should use the keyword **EXPLAIN ANALYZE** before the query:

```
EXPLAIN ANALYZE SELECT * FROM MY_TABLE WHERE NAME = 'Some Name';
```

If an index is not used, the output contains the word **tableScan**:

```
SELECT
    MY_TABLE.ID,
    MY_TABLE.NAME
FROM PUBLIC.MY_TABLE
    /* PUBLIC.MY_TABLE.tableScan */
    /* WHERE NAME = 'Some Name'
    */
    /* scanCount: 42 */
WHERE NAME = 'Some Name'
```

After adding an index to the column **NAME**:

```
CREATE INDEX INDEX1 ON MY_TABLE(NAME);
```

Query execution plan has changed (now the index is applied):

```
SELECT
    MY_TABLE.ID,
    MY_TABLE.NAME
FROM PUBLIC.MY_TABLE
    /* PUBLIC.INDEX1: NAME = 'Some Name' */
    /* WHERE NAME = 'Some Name'
    */
    /* scanCount: 3 */
WHERE NAME = 'Some Name'
```

Usually indexes significantly improve the speed of data retrieval - in my case, the query (because of additional indexes) became faster by 2-3 times.

H2 options

H2 has a number of settings that can improve performance (proposed by default). Almost all available options (except **CACHE_TYPE**) can be found here or by the query:

```
select * from information_schema.settings
```

Some additional parameters can be found there, and also there.

Depending on the option, it can be changed or when connecting to the database, such as:

```
Connection c = DriverManager.getConnection("jdbc:h2:mydatabase;CACHE_SIZE=8192");
```

Or by using the command **SET** (after connection to the database):

```
SET CACHE_SIZE 8192
```

Both ways are available for the most settings.

Cache

As stated in the documentation H2 caches most frequently used data in the main memory. If the cache size is not sufficient to fulfill the request, the same data will be read from disk several times. Therefore the size of cache also affects the first execution of a query (if processing requires reading large enough amount of data from the disk).

The cache size is set by the option CACHE_SIZE in kilobytes and defaults to 16384 (16MB). In my case, this was not enough - increasing the cache twice (up to 32 MB), I got the performance increment in the area of 62%, which is more than 2 times faster.

H2 also allows to choose the caching algorithm: **LRU** (is used by default), **SOFT_LRU** or **TQ**. This can be done using the option **CACHE_TYPE**. As recommended in the documentation, I tried all three algorithms - and it turned out that **TQ** is a bit faster - about 2-4%.

The number of cached queries for the session is specified using the option QUERY_CACHE_SIZE. The default value is 8, which is not much, and probably it will require to increase.

There are a number of options (MAX_MEMORY_ROWS, MAX_MEMORY_UNDO and MAX_OPERATION_MEMORY) related to caching, which can speed up the database, but in my case, the increase in their values did not improve the performance.

MVCC setting

This feature, as described in documentation, provides higher concurrency for modification operations and is disabled by default.

In my case, if you turn off this setting, the application stops working correctly because of the many queries exceeded the timeout.

On the other hand, if there are a small number of concurrent queries, then probably it is not necessary to turn on this option, because when it is turned on, single queries start to run slower around 45%, as shown by my tests.

EARLY_FILTER option

Setting EARLY_FILTER allows table implementations to apply filter conditions early on. By default this option is set to **false**.

After the setting was enabled, the speed of SELECT-queries increased by approximately 53%.

Page and file sizes

As almost all databases, H2 divides memory into the pages. By default page size (option PAGE_SIZE) is equal to 2 kilobytes, which at first glance seems to be insufficient (for instance, in MySQL the default page size is 16 KB), but increasing the **PAGE_SIZE** value (I tried 4, 8, 16 and 32 KB) does not improve performance (and even makes it worse), as well as increases the size of the data files on disk. Thus, the default value of this option unlikely will require changes.

H2 provides possibility to select file system implementation. In some cases, it can also improve performance. For example, under the operating system **Microsoft Windows 7 (x64)** my tests showed that **nio** improves the database performance by 3-4% compared to the default implementation of the file system (**RandomAccessFile**). At the same time, under **Oracle Solaris 11 (x64)** files with random access work as quickly as **nio**.

Yet another opportunity to influence on the database performance is splitting it into multiple files (by default all data is stored in a single file). This can be done by using the following path to the database: **jdbc:h2:split:n:fileName** - then the data will be splitted into the files with size 2^n bytes. My tests showed, that splitting all data on files with size of 16 MB (**jdbc:h2:split:24:fileName**) improved its speed under **Microsoft Windows 7 (x64)** by 4-7%, but under **Oracle Solaris 11 (x64)** on the contrary speed dropped to 5-6% compared to the default setting (when the database is stored in a single file).

Option WRITE_DELAY defines a maximum delay in milliseconds between the end of a transaction and its recording to the disk. By default, its value is 500 ms - and it seems optimal (I tried 200, 1000 and 2000 ms, but the speed of database has not improved).

Conclusion

H2 has enough options to improve its performance. They help to speed up the database by several times. It should be noted that specific values of many options depend on the specifics of the application, used software and hardware - and therefore they should be picked up using appropriate tests.