19th International Conference on Advanced Communication Technology (ICACT 2017)

Server for SQLite Database: Multithreaded HTTP Server with Synchronized Database Access and JSON Data-Interchange

Authors:
Noprianto
Benfano Soewito, Ph.D
Dr. Sani Muhamad Isa
Karto Iskandar
Dr. Ford Lumban Gaol
Dr. Raymond Kosala

Presented by:
Dr. Sani Muhamad Isa

Introduction

- SQLite: serverless database engine that works with single database file
  - No database server to configure
  - Applications: access the database file directly (from multiple operating system processes and/or threads, or with file sharing services)
- How if, applications in mixed environments (mobile, desktop, web-based, or where file sharing is not a reliable option) need to work with a single SQLite database?
  - Replace SQLite with client/server database is not always the best solution (particularly when SQLite is used as main database system, in production)

Server for SQLite

- Communication protocol: HTTP
  - RFC: HTTP/1.0 RFC1945, HTTP/1.1 RFC2616, HTTP/2 RFC7540
  - Has been in use by the World-Wide Web since 1990
  - Virtually everywhere
  - Supported by many programming language runtime environments
- Data-interchange: JSON
  - RFC4627 and ECMA-404 standard
  - Lightweight data-interchange format
  - Libraries available for many programming languages

Database locks

- SQLite locking mechanism
- Multithreaded server: make sure that only one thread can access the database file, at a given time

Query received → acquire lock → perform query → release lock → send response

Server implementation

- Using Python programming language version 3.x
  - Tested in version 3.4
  - Using only standard library (no dependency to external libraries)
  - In single Python script
  - No platform-specific codes
  - Will be modified to make it Python 2.x compatible
- Free/open source software: Source code available from http://noprianto.com

Server implementation: screen shot

```
QwOV8zNjgzKGEpIHZhbHVlcygxOSk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyMCk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyMSk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyMik=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyMyk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyNCk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyNSk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyNik=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyNyk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyOCk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=aW5zZXJ0IGludG8gdGFibGVfMTQ3NzczND
QwOV8zNjgzKGEpIHZhbHVlcygyOSk=&c=1 HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2016 16:46:53] "GET /?q=c2VsZWN0IG1heChST1dJRCkgZnJvbSB0YW
JsZV8xNDc3NzM0NDA5XzM2ODM= HTTP/1.1" 200 -
```

HTTP server is serving GET request to /, with q and c variables.
(Query encoded in Base64)

Testing procedure

- Spawn a number of threads, and each of them will perform a number of database operation (both writing and reading)
- Applied to both empty and non-empty database
- Check if the database is still in healthy state

Testing procedure: screen shot

```
[Thread-16][Table: table_1477734409_8030][29]
[Thread-59][Table: table_1477734409_3683][16]
[Thread-59][Table: table_1477734409_3683][17]
[Thread-59][Table: table_1477734409_3683][18]
[Thread-59][Table: table_1477734409_3683][19]
[Thread-59][Table: table_1477734409_3683][20]
[Thread-59][Table: table_1477734409_3683][21]
[Thread-59][Table: table_1477734409_3683][22]
[Thread-59][Table: table_1477734409_3683][23]
[Thread-59][Table: table_1477734409_3683][24]
[Thread-59][Table: table_1477734409_3683][25]
[Thread-59][Table: table_1477734409_3683][26]
[Thread-59][Table: table_1477734409_3683][27]
[Thread-59][Table: table_1477734409_3683][28]
[Thread-59][Table: table_1477734409_3683][29]
[Thread-1][Table: table_1477734409_7064][response: {'data': [[30]]}]
[Thread-50][Table: table_1477734409_4756][response: {'data': [[30]]}]
[Thread-38][Table: table_1477734409_3077][response: {'data': [[29]]}]
[Thread-25][Table: table_1477734409_9909][response: {'data': [[30]]}]
[Thread-35][Table: table_1477734409_4456][response: {'data': [[30]]}]
[Thread-29][Table: table_1477734409_1602][response: {'data': [[30]]}]
[Thread-9][Table: table_1477734409_1656][response: {'data': [[30]]}]
[Thread-58][Table: table_1477734409_9980][response: {'data': [[30]]}]
```

- Testing is being run, showing response from HTTP server

```
[Thread-984][response: {'data': [['3.8.7.1']]}]
[Thread-986][response: {'data': [['3.8.7.1']]}]
[Thread-985][response: {'data': [['3.8.7.1']]}]
[Thread-989][response: {'data': [['3.8.7.1']]}]
[Thread-988][response: {'data': [['3.8.7.1']]}]
[Thread-987][response: {'data': [['3.8.7.1']]}]
[Thread-993][response: {'data': [['3.8.7.1']]}]
[Thread-991][response: {'data': [['3.8.7.1']]}]
[Thread-990][response: {'data': [['3.8.7.1']]}]
[Thread-992][response: {'data': [['3.8.7.1']]}]
[Thread-994][response: {'data': [['3.8.7.1']]}]
[Thread-995][response: {'data': [['3.8.7.1']]}]
[Thread-997][response: {'data': [['3.8.7.1']]}]
[Thread-996][response: {'data': [['3.8.7.1']]}]
[Thread-998][response: {'data': [['3.8.7.1']]}]
[Thread-999][response: {'data': [['3.8.7.1']]}]
[Thread-1000][response: {'data': [['3.8.7.1']]}]
[Thread-759][response: {'data': [['3.8.7.1']]}]
[Thread-9][response: {'data': [['3.8.7.1']]}]

real    0m1.115s
user    0m0.852s
sys     0m0.252s
```

Response from HTTP server, showing ‘data’ key (if error occurred, ‘error’ key is returned)

Testing procedure: result (1)

Iteration	Number of existing tables	Elapsed time (Real, in second)	Elapsed time (User, in second)	Elapsed time (Sys, in second)
1	0	137.19	1.15	0.36
2	0	137.39	1.15	0.39
3	0	136.55	1.01	0.38
4	60	136.38	1.09	0.42
5	120	140.19	1.17	0.38
6	180	138.90	1.15	0.38

TEST RESULT : 60 THREADS AND 32 QUERIES / THREAD (TOTAL 1920 QUERIES)

Testing procedure: result (2)

Iteration	Number of threads	Number of queries per thread	Elapsed time (Real, in second)	Average time per query	Note
1	100	102	760.46	**0.075**	
2	100	202	1,465.27	**0.073**	
3	> 100	> 200	Canceled	N/A	Multithreading related problem, database not affected

TEST RESULT : NUMBER OF THREADS, NUMBER OF QUERIES PER THREAD, ELAPSED TIME, AND NOTE

Testing procedure: result (3)

Iteration	Number of threads	Elapsed time (Real, in second)	Average time per query
1	1000	1.16	**0.001**
2	1000	1.11	**0.001**
3	2000	4.87	**0.002**
4	2000	2.54	**0.001**

TEST RESULT : NUMBER OF THREADS AND ELAPSED TIME FOR GETTING DATABASE VERSION

Testing procedure: result (4)

Iteration	Number of threads	Number of queries per thread	Elapsed time (Real, in second)	Average time per query
1	2000	2	5.02	**0.001**
2	2000	2	5.10	**0.001**

TEST RESULT : NUMBER OF THREADS AND ELAPSED TIME FOR ERRONEOUS QUERIES

Challenges in current version

- Blocking SQL operation
  - More integration with SQLite is needed, but no definitive proposal yet (if still using HTTP, or HTTP with session)
  - SQLite is fast: For comparison, direct access to SQLite file (without committing change for each query; only at the end) took 50,002 queries in average of total 0.5 second, or 0.00001 second per query
- Cannot guarantee that commands are executed in order they received
  - Proposal: usage of priority queue

Future developments (1)

- Authentication and authorization
  - Limit access to the database by connecting host
  - Use authentication extension (source available)
  - Emulating authentication and authorization feature by using one special table (for example: used by free/open source software SQLiteBoy)
  - Use one special database (then provide a mechanism to allow such meta data to be exported if the database file is moved to another host)

Future developments (2)

- Communication protocol
  - Using HTTP partly because it is ubiquitous
  - There are several database-independent and industry-standard method to work with database systems
  - Most client/server database systems provide their own protocol
- Resource throttling
- Proposed method in this paper is planned to be included in Pangsit, a small domain-specific programming language for data entry

Conclusion

- Based on test results and reliability of SQLite database:
  - Proposed method is applicable
  - During the tests: no database corruption is found
  - Average of elapsed time for SQL queries is also acceptable (in our cases)

Thank you

References (in the paper)

1) The SQLite Development Team. About SQLite. [Online]. Available: https://sqlite.org/about.html
2) The SQLite Development Team. SQLite Frequently Asked Questions. [Online]. Available: https://sqlite.org/faq.html
3) M. Owens, "Embedding an SQL database with SQLite," Linux Journal, 2003(110), 2.
4) The SQLite Development Team. Most Widely Deployed SQL Database Engine. [Online]. Available: https://sqlite.org/mostdeployed.html
5) R. Fielding et al, "Hypertext transfer protocol--HTTP/1.1 (No. RFC 2616)", 1999.
6) N. Nurseitov et al, "Comparison of JSON and XML Data Interchange Formats: A Case Study. Caine", 2009, 157-162.
7) JSON developer. JSON. [Online]. Available: http://www.json.org
8) R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing", 2014.
9) The SQLite Development Team. Implementation Limits For SQLite. [Online]. Available: https://www.sqlite.org/limits.html
10) J. Burnim, and K. Sen, "Asserting and checking determinism for multithreaded programs," In "Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering", 2009, August, pp. 3-12, ACM.
11) The SQLite Development Team. SQLite: Documentation. [Online]. Available: http://www.sqlite.org/src/doc/trunk/ext/userauth/user-auth.txt
12) Noprianto. Simple Web SQLite Manager/Form/Report Application. [Online]. Available: https://github.com/nopri/sqliteboy
13) The PostgreSQL Global Development Group. PostgreSQL: Documentation: devel: Frontend_Backend Protocol. [Online]. Available: https://www.postgresql.org/docs/devel/static/protocol.html
14) Noprianto and B. Soewito, “Pangsit: spreadsheet-like domain-specific programming language for data entry application development,” In “The Eleventh 2016 International Conference on Knowledge, Information and Creativity Support Systems (KICSS)”, 2016.