S. Kozielski et al. (Eds.): BDAS 2017,
CCIS 716, pp. 177 -188, 2017.
DOI: 10.1007/978-3-319-58274-0 15

Interoperable SQLite for a Bare PC

William Thompson, Ramesh Karne, Alexander Wijesinha, and Hojin Chang

Towson University, Towson MD 21252, USA

Abstract. SQLite, a widely used database engine, has been previously transformed to run on a bare PC without the support of any OS or kernel. However, the transformed SQLite database was stored in main memory i.e., it had no file system. This paper extends the transformation process to enable bare PC SQLite to work with standard file system interfaces based on the FAT32 file specification. It further presents mechanisms and programming interfaces for a bare machine file system integrated with SQLite that uses a removable USB flash drive. The bare SQLite database and file system can interoperate with conventional OS-based database systems. It can be adapted in the future to work with bare Web browsers, large bare databases, other bare applications, and bare mobile devices.

1 INTRODUCTION

SQLite is a self-contained, zero-configuration, stand-alone (not client/server) lean database management system. It is commonly used in Web browsers, mobile devices and embedded systems. A SQLite amalgamation [19] consisting of about 130K lines of code has been transformed to run on a bare PC with no operating system (OS) or kernel [12][13]. The bare PC SQLite version uses the :memory option, where the database is stored in real memory with no standard file interfaces. Potential advantages of running SQLite or applications such as Web servers or VoIP clients on a bare PC include the elimination of OS overhead and OS-related vulnerabilities.

This paper discusses the addition of a standard FAT32 file system [11] to the bare PC SQLite version so that it can interoperate with a conventional OS-based SQLite database engine. The lean bare PC FAT32 file system [10], which is intertwined with the associated application, uses a removable USB flash drive.

2 BARE MACHINE COMPUTING

Approaches to reduce OS overhead and improve application performance include Exokernel [2], OS-Kit [14] and Palacio/Kitten [9].In a bare machine computing or bare PC application, all intermediary software in the form of an OS, lean kernel, or external libraries is eliminated. Bare applications are written in single programming language such as C/C++ and directly communicate to hardware without middleware or a centralized kernel [6][7]. Bare PC Web servers [4], Webmail servers [1], email servers [3] and split-servers [16][17] have been developed previously.

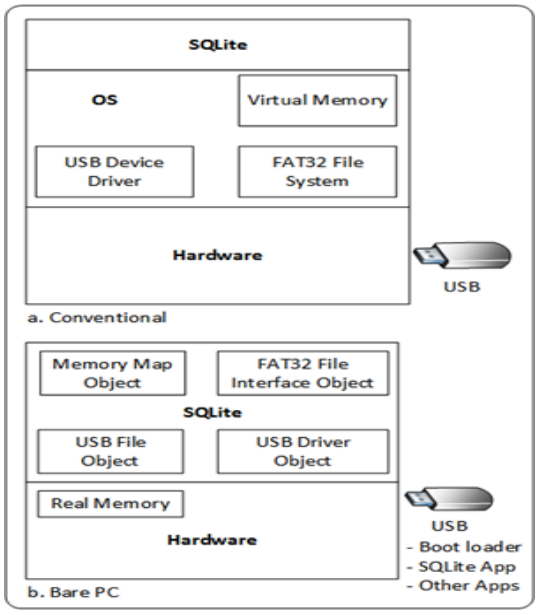

Fig. 1. Software Architecture

3 SQLITE TRANSFORMATION

The SQLite amalgamation package used in the transformation [13] has two source files (shell.c and sqlite3.c) and two header files. It runs on Windows/Visual Studio (VS). Detailed documentation on SQLite amalgamation is given in [20]. SQLite provides a command line interface and a file interface for user input. SQL queries can be run in single command mode or as a transaction. The database is stored using a standard SQLite file format in Windows or Linux. The transformation process is described in detail in [12][13]. It eliminated 85 system calls and replaced them with direct bare PC hardware interfaces. These system calls can be classified as file, timer, data types, process, memory, and standard I/O. The file system calls were not replaced as the database was intended to run in main memory. The remaining calls were replaced with equivalent bare PC calls, which are much simpler and do not require any centralized OS or kernel. These calls run in single user mode along with its application. Conventional database management systems use standard file systems that are provided by the host OS, and it is possible to port a database from one OS platform to another by using middleware tools [18]. However, these tools are themselves platform dependent and cannot be used to adapt an OS-based database/file system to run on a bare PC.

4 DESIGN AND INTERFACES

Figure 1 shows respective architectural views of conventional and bare PC environments for running SQLite. We assume USB mass storage is used to store the SQLite database file based on the FAT32 file system. In a conventional environment such as Visual Studio (VS) on Windows, SQLite runs on top of the OS, which provides the necessary interfaces for virtual memory, file management and device drivers. In a bare PC, four objects MemObj, FileObj, UsbFileObj and UsbObj provide the complete functionality needed for the operation of SQLite. The MemObj provides real memory allocation and deallocation needed for SQLite (called by malloc() and free()). The FileObj provides the above file API. A USB

```
struct sqlite3_vfs {
   int iVersion;                /* Structure version
number (currently 3) */
   int szOsFile;                /* Size of subclassed
sqlite3_file */
   int mxPathname;              /* Maximum file
pathname length */
   sqlite3_vfs *pNext;          /* Next registered
VFS */
   const char *zName;           /* Name of this
virtual file system */
   void *pAppData;              /* Pointer to
application-specific data */
   int (*xOpen)(sqlite3_vfs*, const char *zName,
sqlite3_file*, int flags, int *pOutFlags);
   int (*xDelete)(sqlite3_vfs*, const char *zName,
int syncDir);
   int (*xAccess)(sqlite3_vfs*, const char *zName,
int flags, int *);
   int (*xFullPathname)(sqlite3_vfs*, const char
*zName, int nOut, char *zOut);
   void *(*xDlOpen)(sqlite3_vfs*, const char
*zFilename);
   ...
   int (*xCurrentTime)(sqlite3_vfs*, double*);
   int (*xGetLastError)(sqlite3_vfs*, int, char *);
   int (*xCurrentTimeInt64)(sqlite3_vfs*,
sqlite3_int64*);
   int (*xSetSystemCall)(sqlite3_vfs*, const char
*zName, sqlite3_syscall_ptr);
   sqlite3_syscall_ptr
(*xGetSystemCall)(sqlite3_vfs*, const char
```

Fig. 2. SQLite virtural file system object

file object (UsbFileObj) provides initialization, reset and plug-and-play features

for all USB ports. Finally, a USB device driver object (UsbObj) provides driver functionality [8] specific to bare PC applications, which uses USB 2.0 standard specification [15], enhanced host controller specification [5], and USB mass storage specification [21]. These four objects are an integral part of SQLite and a database application running on a bare PC.

We enhanced the bare PC file system in [10] to be fully compatible with the

Table 1. Attributes for $sqlite3\_vfs$

Windows	**Bare PC**
iVersion	iVersion -l
sizeof(winFile)	sizeof(bareFile)
MAX_PATH	$MAXPATHNAME$ −512
pNext	pNext −0
win32	”bare”
pAppData	pAppData - 0
winOpen	bareOpen
winDelete	bareDelete
winAccess	bareAccess
winFullPathname stub	only
winDlOpen	stub only
winDlError	stub only
winDlSym	stub only
winDlClose	stub only
winRandomness	bareRandomness
winSleep	bareSleep
winCurrentTime	bareCurrentTime
winGetLastError	N/A (optional)
winCurrentTimeInt64	N/A
xSetSystemCall	N/A
xGetSystemCall	N/A
xNextSystemCall	N/A

FAT32 specification so that it can interoperate with any OS platform. The enhanced file API has five functions, namely: createFile(), deleteFile(), resizeFile(), flushFile(), and flushAll(). These are used to interface with the SQLite database. The createFile() function has a file name, memory address pointer, file size and file attributes. It returns a file handle. The file handle is the index value of the file in a file table structure, which has all the control information of a file. This approach enables a direct index into the file table to be used without the need for searching. The deleteFile() function uses the file handle to delete a file. The resizeFile() function is used to increase or decrease a previously allocated file size. The flushFile() function updates the USB mass storage device from its related data structures and memory. The flushAll() interface is used to flush all files and related structures onto the USB drive. A single executable includes all

the bare PC code. SQLite runs as a separate task within the application. As SQLite is written in C, the code is wrapped in C++ to communicate with the object-oriented code in the bare PC. In a Windows environment, the USB contains the SQLite database; in a bare PC, it contains the SQLite database along with boot, load, SQLite application, and other applications such as a Web server if needed.

SQLite provides a separate wrapper for a given OS interface or virtual file system (VFS) [20]. This wrapper approach in SQLite motivated us to develop a bare PC API that substitutes for a given OS and enables the development of a bare PC file system interface to SQLite. The three structures changed in the SQLite code are $sqlite3\_vfs$, $sqlite3\_io\_methods$, and $sqlite3\_file$. A brief overview of the changes is given below.

4.1 Virtual File System Object

The $sqlite3\_vfs$ (virtual file system) structure shown in Figure 2 defines the interface between the SQLite core and the underlying OS [20]. In the Windows OS, an instance of this structure illustrates the attributes needed for SQLite as shown in the left column of Table 1. A total of 22 functions are used in this object. The equivalent bare PC functions are shown in the right column of this table. For the bare PC implementation, five functions are optional and not applicable, and five methods are provided with stubs as these are Windows API calls. The remaining functions are implemented for the bare PC. The most important method in this object is bareOpen, which is used to open/create a SQLite database file. In order to substitute our function, we created a $register\_barevfs()$ function and inserted it into the $sqlite3\_os\_init()$ function, which initializes all OS parameters. Figure 3 illustrates a trace of SQLite control flow from the $open\_db()$ call to the bareOpen() function.

4.2 I/O Method Object

SQLite manipulates the contents of the file system using a combination of four types of file operations: create, delete, truncate and write. The SQLite object named $sqlite3\_io\_method$ consists of I/O functions as shown in the left column of Table 2. There are 17 functions in the Windows API related to the SQLite VFS. Eight functions are not applicable to a bare PC environment and the other nine are implemented for bare PC applications as shown in the table.

The bareRead() and bareWrite() functions do read and write respectively using real memory, and all the USB structures and data are memory mapped into main memory. When the bareSync() function is called, the bare PC system flushes the database file to the USB. In addition, it also flushes the root directory, modified FAT tables and appropriate file data. The bare PC application directly invokes the device driver to read or write to a USB flash drive. The flush operation is only used when needed or when a transaction is complete. The frequency of updates to mass storage is controlled by the application. The bare PC file system is

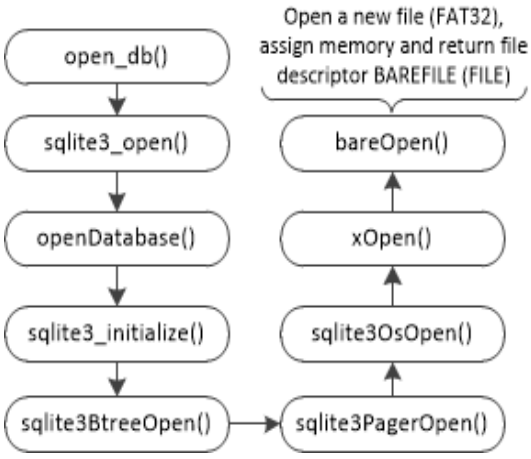

Fig. 3. Trace of SQLite control flow

designed for optimal performance and reduces write operations as they are slow in a USB.

4.3 File Object

The file object structure represents an open file in the SQLite OS interface layer. We have extended this object in bareFile. The extended structure consists of the $\_iobuf$ structure which contains parameters that are needed to implement the bare PC file system. For example, cacheStartAddr is the real memory address provided by the memory object. The index value points to the entry in the file table. The openFile method also has an instance of a bareFile which is a $sqlite3\_file$ type. This bare PC file instance is linked with bareio which points to all the functions needed in a bare PC. These are the $sqlite3\_io\_methods$ described in the previous section. The implementation of the above functions is done in C. The size of the new code is approximately 1,300 lines including comments. The new SQLite runs on a bare PC FAT32 file system.

5 INTEROPERABILITY AND PERFORMANCE

The SQLite database file that runs on a bare PC is interoperable with one running on VS/Windows. Thus, a database can be created in Windows or on a bare PC and used in either environment. The same can be done with data updates. In conventional systems, interoperability is achieved by porting the database management system to run on a different platform. For example, an Oracle DBMS running on Linux can be ported to run on Windows. While such database systems have OS-specific dependencies (e.g. Oracle Linux, Oracle Solaris, Oracle Windows), the bare PC SQLite database system is independent of

Table 2. I/O methods for *sqlite3_file*

Windows	**Bare PC**
iVersion	iVersion -l
winSync	bareSync
winFileSize	bareFileSize
winLock bareLock	N/A
winUnlock	bareUnlock -N/A
winCheckReservedLock	bareCheckReservedLock -N/A
winFileControl	bareFileControl -N/A
winSectorSize	bareSectorSize
winDeviceCharacteristics	bareDeviceCharacteristics
winShmMap	N/A (optional)
winShmLock	N/A
winShmBarrier	N/A
winShmUnmap	N/A

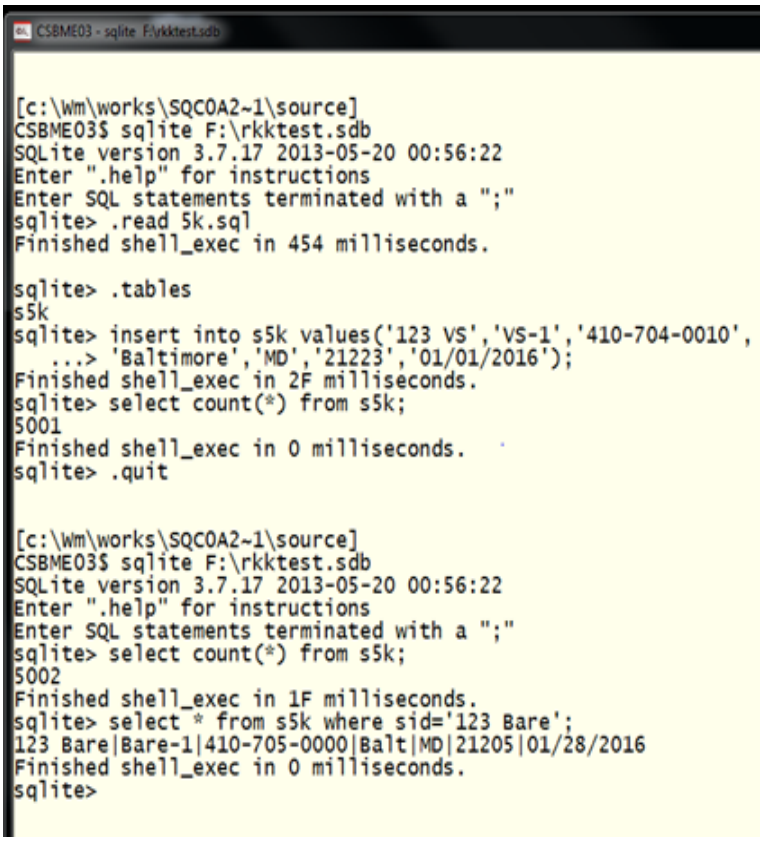

Fig. 4. SQLite on Windows

any OS platform; it can run on any x86 based machine with a USB 2.0 interface and create a database file (FAT32 format) that can be used by SQLite running on any platform. Also, one can easily design and implement interfaces that work with other file system formats for different computer hardware architectures.

5.1 Interoperability

This section describes experiments to demonstrate the interoperability of SQLite. The measurements were conducted on Dell Optiplex 960 models with a 3.16 GHz dual-core system. However, we only ran this on a single core processor to compare with a single core bare PC application. The SQLite database was run on Windows 7 with Visual Studio (VS) 2010 and also on a bare PC with no OS or hard disk. The USB flash drive contains the bare application suite including boot and load programs. We also ran SQLite and a Web server application in a multi-threaded manner on the bare PC.

Initially, a USB is created with the bare application suite and no database file on it. This is a bootable and executable USB for the bare PC (it contains bare boot sector, loader, and application). The same USB is then used to save a SQLite database file created by VS in Windows. Figure 4 shows a screenshot on the Windows machine, where the database was created and tested. Here, one table (name: s5k) containing 5,000 inserts was edited in a file (5k.sql) formed as a single transaction. This file was read by SQLite using the .read command and executed. The .tables command shows the name of the table (s5k) in VS. One new record was inserted into the above table (123 VS) and the count(*) SQL statement shows 5,001 records in the database. At this point, we use .quit from VS and obtained the created database rkktest.sdb, which was saved in the USB.

This USB with the database file is used to boot the bare PC. The bare PC, after initialization and loading, reads the database file into memory as a memory mapped file. It recognizes the existing database which came from VS/Windows environment. Figure 5 shows a bare PC screenshot showing the newly inserted record in VS (123 VS). Now, we have inserted a new record 123 Bare in the bare PC database. The new count(*) shows 5,002 records indicating that it loaded the database successfully and added a new record. Figure 6 shows the database activities performed in the bare PC; at the end it flushes the database.

This bare PC database is used in VS/Windows. As shown in Figure 4, the count(*) shows 5,002 records and the 123 Bare record. The above tests show that bare SQLite is interoperable with conventional OS-based SQLite in addition to having the same basic management and storage capabilities for data and files. While SQLite interoperability is easily provided by conventional systems with OS support, we have shown that an interoperable database can run on a bare PC with no OS or kernel overhead.

5.2 Performance

This section shows some basic performance data collected to illustrate the potential gains due to eliminating OS overhead in bare SQLite. The database queries

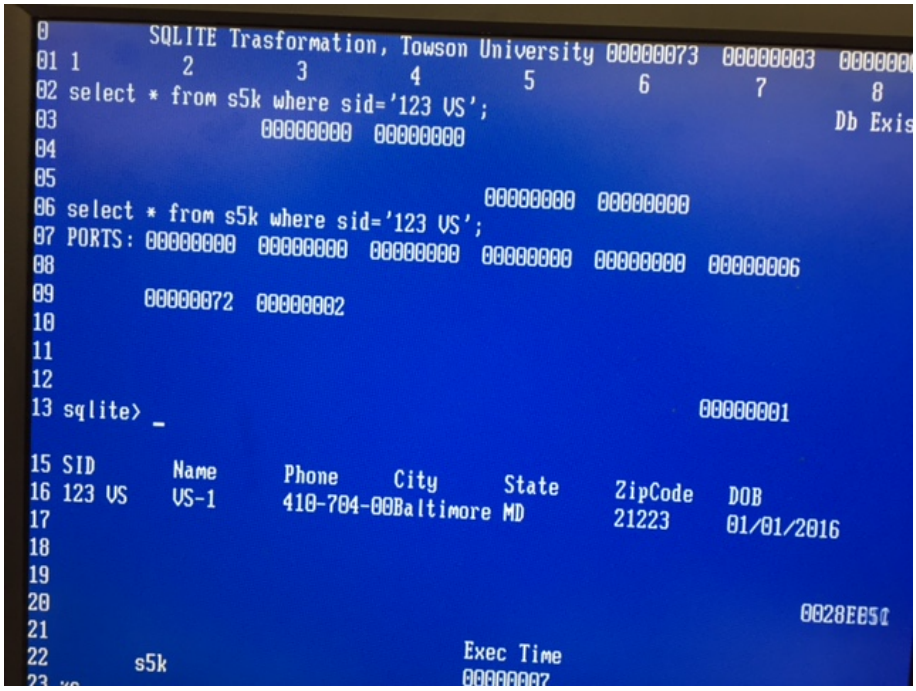

Fig. 5. Bare PC: read database created in VS

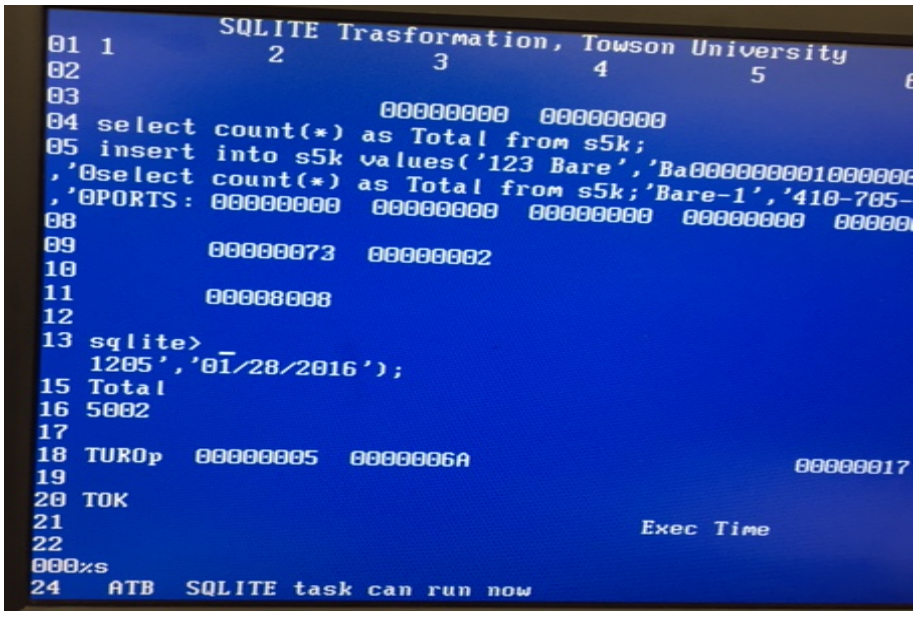

Fig. 6. Bare PC: inserting one record

in this study were done by using a single SQL statement (one at a time), or by collecting a set of SQL statements in a single transaction using BEGIN and COMMIT. Figure 7 shows the performance when the number of inserts into a

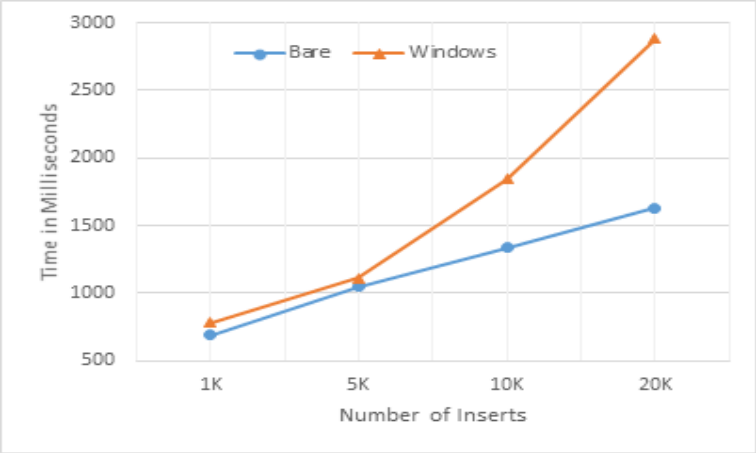

Fig. 7. Inserts with transactions

single table is varied from 1,000 to 20,000 records. These inserts were placed in a file and run by using the .read meta-command in a single transaction. SQLite on a bare PC SQLite performs much better than on VS/Windows as expected. The bare PC performance improvements are attributed to less overhead in the hardware interfaces compared to system calls in OS. Figure 8 shows run times for inserting records without transactions when the number of inserts is varied from 10 to 100. Notice that the run times for Windows are very large as SQLite creates and updates a journal file for each SQL insert statement. In transaction mode (Figure 7), the times for Windows were less because SQLite only flushes the file at the end of a transaction. Without transactions (Figure 8), the bare PC system performs much faster than Windows for individual SQL statements as it updates the main database and journal in memory and does a final flush after running all the queries from a given file.

To make a fair performance comparison between Windows and bare SQLite without transactions, a future study will need to run the above experiment when both systems use the journal file approach and when both do not. The journal file approach provides more reliability as it provides frequent updates, but requires more time to run. Note that the journal files are accessed multiple times depending on the number of SQL statements or transactions.

6 CONCLUSION

We described a SQLite database with a file system that runs on a bare PC and is interoperable with OS-based systems. The design and implementation details were provided to show how the bare PC file system interfaces to the SQLite virtual file system. We also tested interoperability of the bare SQLite system with a conventional Windows SQLite system and showed how a VS/Windows database can be used in a bare PC environment and vice versa. The performance results suggest the feasibility of building scalable bare database systems. A similar approach can be used to transform other OS-based databases such

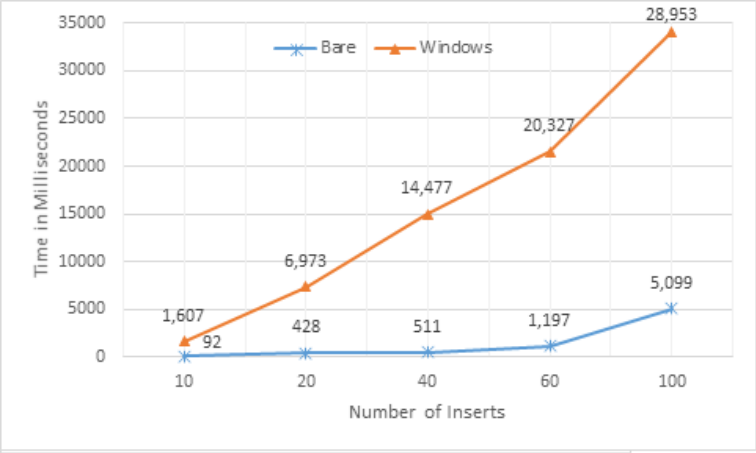

Fig. 8. Inserts without transactions

as MySQL and PostgreSQL server to run on bare machines. Also, the database management functions can be split into a user interface and a database execution engine. This will enable standard and familiar user interfaces to be used with a secure bare database engine that can be hidden behind a conventional server. For example, one could use bare PC SQLite to create and manage databases, and use the Windows OS to provide user interfaces. Database (and other) servers are naturally suited for bare PC or bare machine applications as they focus on a single monolithic executable and are easily tailored for the backend. Future studies could investigate the pros and cons of splitting database management functions in this manner.

References

[1] P. Appiah-Kubi, R. K. Karne, and A. L. Wijesinha. The Design and Performance of a Bare PC Webmail Server. The 12th IEEE International Conference on High Performance Computing and Communications, AHPCC 2010, Sept 1-3, 2010, Melbourne, Australia, pp. 521-526

[2] D. R. Engler and M.F. Kaashoek. Exterminate all operating system abstractions. Fifth Workshop on Hot Topics in Operating Systems, USENIX, 1995, p. 78.
[3] G. H. Ford, R. K. Karne, A. L. Wijesinha, and P. Appiah-Kubi. The design and implementation of a bare PC email server. 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC), 2009, pp. 480-485.
[4] L. He, R. K. Karne, and A. L. Wijesinha. The design and performance of a bare PC Web server. International Journal of Computers and Their Applications, IJCA, Vol. 15, No. 2, June 2008, pp. 100-112.
[5] Intel Corporation. Enhanced Host Controller Interface specification for Universal Serial Bus. March 2002, Rev 1, http://www.intel.com/technology/usb/download/ehci-r10.pdf.
[6] R. K. Karne, K. V. Jaganathan, N. Rosa, and T. Ahmed. DOSC: Dispersed Operating System Computing. 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2005, pp. 55-61.
[7] R. K. Karne, K. V. Jaganathan, and T. Ahmed. How to run C++ applications on a bare PC. SNPD 2005, Proceedings of SNPD 2005, 6th ACIS International Conference, IEEE, May 2005, pp. 50-55.
[8] R. K. Karne, S. Liang, A. L. Wijesinha, and P. Appiah-Kubi. A bare PC mass storage USB device driver. International Journal of Computers and Their Applications, Vol. 20, No. 1, March 2013, pp. 32-45.
[9] J. Lange, et. al. Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing. 24th IEEE International Parallel and Distributed Processing Symposium, Apr. 2010.
[10] S. Liang, R. K. Karne, and A. L. Wijesinha. A Lean USB File System For Bare Machine Applications. The Proceedings of the 21st International Conference on Software Engineering and Data Engineering, ISCA, June 2012, pp.191-196.
[11] Microsoft Corp. FAT32 file system specification. http://microsoft.com/whdc/system/platform/firmware/fatgn.rnspx, 2000.
[12] U. Okafor, R. Karne, A. Wijesinha, and P. Appiah-Kubi. A Methodology to Transform an OS-Based Application to a Bare Machine Application. The 12th IEEE International Conference on Ubiquitous Computing and Communications (IUCC-2013), July 16 - 18, Melbourne, Australia, 2013.
[13] U. Okafor, R. K. Karne, A. L. Wijesinha and B. Rawal. Transforming SQLITE to Run on a Bare PC. In Proceedings of the 7th International Conference on Software Paradigm Trends, pages 311-314, Rome, Italy, July 2012.
[14] The OS Kit Project. School of Computing, University of Utah, Salt Lake, UT, June 2002, http://www.cs.utah.edu/flux/oskit.
[15] Perisoft Corp. Universal Serial Bus Specification 2.0. *http : //www.perisoft.net/engineer/usb_20.pdf*.
[16] B. Rawal, R. Karne, and A. L. Wijesinha. Splitting HTTP requests on two servers. 3rd Conference on Communication Systems and Networks (COMSNETS), 2011.
[17] B. Rawal, R. K. Karne, and A. L. Wijesinha. Mini Web server clusters for HTTP request splitting. IEEE International Conference on High Performance, Computing and Communications (HPCC), 2011, pp. 94-100.
[18] F. F. Rezende and K. Hergula. The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateway. International Conference on Very Large Databases (VLDB 98), 1998, pp. 146-157.
[19] SQLite. *http : //www.sqlite.org/download.html*.
[20] The SQLite OS Interface or VFS. *http : //www.sqlite.org/vfs.html*.
[21] Universal Serial Bus Mass Storage Class, Bulk Only Transport, Revision 1.0, 1999. http://www.usb.org.