WHITE PAPER

Performance Metrics & Benchmarks: Berkeley DB

SLEEPYCAT SOFTWARE

Makers of Berkeley DB

Purpose

This white paper presents performance measurements for Berkeley DB under various conditions and on several major operating systems and platforms. Performance estimates presented here are designed to help you understand what to expect from Berkeley DB in some common configurations on commodity hardware running widely used operating systems. Your application performance depends on your data, data access patterns, cache size, other configuration parameters, operating system and hardware. Benchmarks are almost never representative of the performance of a particular application; however, they provide some guidelines and help to set basic operational expectations.

Introduction

Sleepycat Software's Berkeley DB is a database engine designed to serve as the substrate within any application requiring efficient and robust data storage. Berkeley DB is a complete data management library, including support for concurrent access, ACID transactions and recovery, and replication for highly available systems. Berkeley DB is designed to support any structured information model and has been incorporated into relational, object, network and hierarchical systems. Due to its scalability and minimal overhead, it has also been used in a number of reliable messaging, data streaming, caching and data archival systems.

Platforms and Testing Methodology

All performance tests were run on the following commodity hardware and operating system combinations, without customized configuration, tuning or any other performance enhancing techniques. All tests were run on clean installations in multi-user mode systems that were quiescent. For the workloads presented, these are the results that can be obtained under actual use.

The following platforms represent the most common operating systems and hardware platforms we've encountered in our existing customer base. It is by no means an exhaustive list, nor does it represent the extent of supported platforms for Berkeley DB. The version of Berkeley DB tested was 4.3.

- Linux – SuSE Linux 9.1 running on an AMD Athlon™ 64 processor 3200+ at 1GHz system with 1GB of RAM. Western Digital Caviar SE 120GB EIDE (7200RPM) with 8MB cache RAM was connected via an SIS 5513 controller formatted with the EXT3 file system with journaling enabled. The compiler was GCC 3.3.3. The system reported 4046.84 BogoMIPS.
- Solaris – Sun Solaris 10 (x64) running on a SunFire v20z with dual AMD Opteron™ processors at 2GHz with 16GB of RAM. A 80GB Seagate Cheetah (ST373453LC) SCSI hard drive was connected via an integrated SCSI controller on a PCI-X bus and formatted with the UFS filesystem.
- Windows XP – Microsoft Windows™ XP Professional (XP Version 2002, Service Pack 2) running on an AMD Athlon™ 2600+ at 2.0GHz with 768MB RAM. A DiamondMax Plus9 (6Y060L0) 60GB(7200RPM) DMA/ATA-133 hard drive with 2MB cache RAM was connected via an integrated nVIDIA nForce2 SATA controller into the PCI bus and formatted with an NTFS file system with journaling enabled. The system was compiled with Microsoft Visual Studio 6.
  (NOTE: The timer resolution of Windows XP is much lower than that provided by the other operating systems. As a result the results may be skewed up to 10%.)
- BSD Unix – FreeBSD 5.3p4 running on an Intel™ Pentium™ 4 CPU at 3.00GHz with 2GB of RAM. An integrated Intel ICH5 SATA150 controller on a PCI bus controlled a Barracuda 7200.7 Plus (ST3200822A) Ultra ATA/100 (7200RPM) hard drive formatted with the UFS file system. The compiler was GCC 3.4.2.
- Mac OS/X – Mac OS/X version 10.3.9 running on a dual Power Mac G4 CPUs at 1.25GHz with 2GB of DDR SDRAM. IBM Deskstar 120GXP 120GB UATA100 hard drive was connected via an integrated ATA bus formatted with the HFS+ file system with journaling enabled. The compiler was GCC 3.3.

Note that the systems vary greatly. This is intentional. It is not the goal of this paper to demonstrate the relative performance of the profiled systems when performing similar tasks, but rather to demonstrate basic performance metrics on a wide variety of systems so that you can estimate how your system might perform when using Berkeley DB.

Any transactional database claiming durability (the 'D' in ACID) will necessarily have to interact with some form of stable storage or replicate the transaction to other systems. The most common case is to use a hard disk drive. Hard drive technology, bus subsystems, and memory interconnects vary widely, as do their performance. For this set of tests, we used commodity disk technology – that is, slow disks with average subsystems. If you add a fast disk subsystem with the fastest disks, you'll see much higher throughput with Berkeley DB, just as you would with any disk-bound application. Additional hardware options can also affect performance. For instance, using a disk subsystem with a battery-backed memory write cache will improve throughput. There are other, faster storage systems, some based entirely on memory, but they are less common. These hardware-based techniques affect all databases in a similar manner and so

are irrelevant for this study. This paper explores the performance curve of the Berkeley DB software, not the surrounding hardware.

Download the raw results at http://downloads.sleepycat.com/performance-results-db-4.3.txt
Download the test code at http://downloads.sleepycat.com/perf.zip

Performance

Throughput – records read or written in a fixed time interval – is the most common measure of database performance. Each test in this benchmark uses fixed-size records containing keys that are eight bytes long and a data portion of thirty-two bytes. Pages are 32KB, the buffer for bulk transfers was set to 4MB, the log buffer is 8MB, and the buffer cache is 32MB. Source code for the benchmark is roughly 350 lines of C code and may be downloaded from the URL supplied above.

Berkeley DB supports concurrent database access on multi-processor systems. Many application workloads will benefit from such concurrency. Regardless of the system tested, the effects of concurrency are not measured in the following tests. The tests are single-threaded and single-process. On a multi-processor system, they will use no more than one of the CPUs at a time.

This first test measures throughput as operations per second using Berkeley DB Data Store (DS). Since this is a non-transactional, in-memory test, performance is dependent on the operating system, memory, and CPU, and not the underlying I/O or disk system for the read tests. When writing using DS, the data is cached in memory and flushed when buffers are filled or when the application explicitly calls for data to be flushed to disk. These tests are written such that writes occur in memory and are not flushed to disk.

DS (ops/sec)	Linux	Solaris	Win XP	BSD	Mac OS/X
Single-record read	1,002,200	1,008,580	1,000,000	1,108,920	524,360
Single-record write	766,034	550,748	447,628	614,116	317,141

Berkeley DB provides optimized access to sequential data allowing for faster overall throughput. If you read large amounts of data sequentially from your databases, using the bulk access APIs will improve your read performance.

DS (ops/sec)	Linux	Solaris	Win XP	BSD	Mac OS/X
Read using bulk APIs	13,501,800	12,114,500	4,565,910	19,299,700	5,615,280
Speedup relative to non-bulk APIs	13.47	12.01	4.57	17.40	10.71

Most applications using Berkeley DB will require transactional access to data as a result of concurrent access or simply to maintain a reliable repository of data. This is accomplished by using the Berkeley DB Transactional Data Store (TDS) features. In this first test of TDS we enable the locking, logging and transaction subsystems. Although read-only transactions do not perform logging, performing reads in transactions will introduce locking overhead. That overhead is reflected in the following test results.

TDS (ops/sec)	Linux	Solaris	Win XP	BSD	Mac OS/X
Single-record read	466,623	328,809	259,134	331,523	171,549
Relative to DS reads	.609	.597	.579	.540	.541

The transactional system can be configured to run entirely in memory. If the process fails, the system is powered down, or some other unforeseen failure occurs, the data will be lost. Essentially, durability is sacrificed for speed. Replicated systems running entirely in core memory using Berkeley DB High Availability (HA) can provide full ACID transactional semantics, including durability, as long as one replica remains active, and will perform much faster than systems storing data to disk. The following measures only the in-memory speed of the transactional system without replication.

TDS in-memory(ops/sec)	Linux	Solaris	Win XP	BSD	Mac OS/X
Single-record write	239,824	143,466	125,486	111,671	82,256
Relative to DS writes	.313	.260	.280	.182	.259

When designing your system, your requirements will determine what, if any, transactional guarantees you can relax. The following tests examine the performance of each tradeoff from the most relaxed to the most stringent.

Applications requiring the 'ACI' (atomic, consistent, and isolatable) characteristics of transactions but willing to risk the 'D' (durability) can configure Berkeley DB such that it does not synchronize transaction logs to disk on commit. In this case, the transaction log buffers are maintained in memory and only written to stable storage when Berkeley DB determines that they are full, rather than as a component of the transaction. If the application, operating system, or hardware should fail, the data in those buffers that is not yet written to disk is lost. This 'TDS no-sync' case is measured below.

TDS no-sync (ops/sec)	Linux	Solaris	Win XP	BSD	Mac OS/X
Single-record write	141,109	107,626	68,521	70,582	68,591
Relative to TDS in-memory logs	. 588	.755	.546	.632	.834

The performance is nearly identical to that of the 'TDS in-memory logs' case. In both cases, the disk is never accessed during the test. The difference is that for the 'TDS no-sync' case, the data is eventually written to disk.

Next, we consider a configuration that will guarantee no loss of data after an application crash, as long as the operating system does not also crash. TDS is again configured with locking, logging and transactions all enabled. However, in this case Berkeley DB requests that the operating system write to disk, but not explicitly flush those writes to disk at each transaction commit. The operating system will acknowledge the write and is likely to buffer writes for some amount of time before flushing the data to disk. In this mode, TDS is sacrificing some durability, but less than before.

TDS write no-sync (ops/sec)	Linux	Solaris	Win XP	BSD	Mac OS/X
Single-record write	90,774	85,101	45,748	62,087	34,888
Relative to TDS no-sync	.643	.791	.668	.880	.509

Notice the significant drop in performance, as each transaction commit requires a call into the operating system and copies data from the application's memory to the operating system's write buffers.

If we enable fully synchronous writes, throughput will be limited by your disk's ability to perform such writes. The resulting numbers are, therefore, completely determined by your disk system and not the software.

Note that it is possible to get spectacular numbers in this mode, because some disks report writes as having completed once the data has been written into the disk's cache, instead of when the data is actually on the disk. This performance boost comes at the price of durability guarantees, and we advise running critical applications on disk subsystems that do not respond to synchronous write requests until the data is persistently on the disk.

Berkeley DB can optionally group commit operations, allowing higher throughput when doing synchronous transactions.

In conclusion, Berkeley DB allows application designers to make transaction durability tradeoffs when building a high throughput system, based on their design goals and the characteristics of the data involved. With an understanding of these and other tuning parameters found in Berkeley DB and in your deployment system it is possible to exceed these measurements.

Aspects of Berkeley DB Not Measured

- Out of cache performance – The impact of storage to disk is so great that when measuring database performance out of cache, you are essentially measuring the underlying operating system and associated hardware. The time spent waiting on disk accesses takes up the vast majority of the overall access time. As a result, the tests have been designed to work entirely in cache except for those times when it is necessary to write to disk for transactional guarantees. Systems with faster disks or disk controllers with built-in RAM-based cache will perform better.

- Recovery time – Transactional databases must be able to recover to a known state at startup. Berkeley DB, like most transactional database systems, will recover in time proportional to the amount of data written to the log since the last checkpoint. The additional cost of taking the checkpoints is not reflected in these measurements.
- Deadlock discovery and resolution time – Transactional databases can deadlock. Berkeley DB offers many different methods for deadlock detection and resolution. These benchmarks do not measure the cost of deadlock processing as they are written to be deadlock-free.
- Startup time – A database's cache subsystem is one of the most performance-critical aspects of its design. There are two states for a cache: empty and warm. We do not measure the cost of warming the cache. All benchmark programs first warmed the cache and then began collecting timings.
- Effects of concurrency – Multi-threaded applications often get better performance than single-threaded applications, because whenever any single thread blocks for I/O, others can use the CPU to do useful work. This is especially true in Berkeley DB, since the database library includes support for a feature called "group commit," which allows one active thread to do work on behalf of other threads when one or more transactions commit. Concurrency also introduces opportunities for contention for locks, buffers or other resources. In general, multi-threaded applications see performance improvements until contention among too many threads causes performance to fall off.
- Access Methods – Berkeley DB provides a few different access methods. Each method provides a different performance characteristic. The measurements in this paper use the btree access method because it is the fastest access method for a wide range of workloads and data sizes.
- HA – The Berkeley DB HA subsystem is a single master, multi-replica system that supports fail-over and master election using a Paxos-compliant election algorithm. It replicates transaction logs among systems. The transport of these logs is application-specific. To replicate database information, Berkeley DB HA uses a transport layer provided by the application to relay coordination messages and log segments among replicas. Any form of message transport can be employed. The application designer may choose among hardware-specific transport, TCP/IP or any other communications infrastructure. For this reason, measurement of the HA system is beyond the scope of this paper.

Comparing Berkeley DB to Relational Database Management Systems (RDBMS)

Client/Server architecture increases complexity and slows down performance because applications must cross a process boundary and often a network link to access data. SQL query processing is extra code that a relational engine must execute at runtime. Parsing, planning, optimizing and executing a query introduce runtime overhead to each data access. Applications that need the full expressive power and flexibility of a SQL engine can choose from a large number of excellent relational engines. If you need a fast, reliable, embeddable data management engine without a human database administrator, then Berkeley DB is a more attractive choice. For applications of this sort, Berkeley DB can deliver much better performance than a client/server, relational engine. Berkeley DB delivers the most direct route to data and avoids cross-process copies, query parsing and planning overhead found in relational systems.

Naturally, head-to-head comparisons depend heavily on the application and its data management requirements. Sleepycat customers who have switched from client/server, relational products to Berkeley DB for storage have reported dramatic performance increases, ranging from several times faster to several orders of magnitude faster.

Conclusions

Berkeley DB, as a component of your application, can be configured in a number of different ways to meet specific application requirements for speed, scale, and reliability. This paper has taken a look at a few of the critical aspects of any database system and shown that Berkeley DB has benefited from years of research, design and careful tuning.

Berkeley DB is the most efficient, most scalable, and fastest database engine available today. Berkeley DB offers extraordinary programming flexibility for applications ranging from mobile phones to desktop applications, large servers, networking gear and major websites. Download Berkeley DB today at http://www.sleepycat.com/products/db.shtml, and run the performance tests on your platform. If you need assistance, Sleepycat Software offers technical support, on-site training and consulting services, including performance evaluation and tuning conducted by one of our performance engineers. Contact info@sleepycat.com for information.

Sleepycat Software www.sleepycat.com makes Berkeley DB, the most widely used open source developer database in the world with over 200 million deployments. Customers such as Amazon.com, AOL, Cisco Systems, EMC, Google, Hitachi, HP, Motorola, RSA Security, Sun Microsystems, TIBCO and VERITAS also rely on Berkeley DB for fast, scalable, reliable and cost-effective data management for their mission-critical applications. Profitable since it was founded in 1996, Sleepycat is a privately held company with offices in California, Massachusetts and the United Kingdom.

For further information, please contact Sleepycat by sending email to info@sleepycat.com or visiting Sleepycat's website at www.sleepycat.com.

SLEEPYCAT SOFTWARE
Makers of Berkeley DB

Corporate Headquarters
Sleepycat Software Inc.
118 Tower Road
Lincoln, MA 01773
USA

West Coast Office
Sleepycat Software Inc.
5858 Horton St. Suite 265
Emeryville, CA 94608
USA

European Office
Sleepycat Europe Ltd.
Coronation House,
Guildford Rd.
Woking, GU22 7QD
United Kingdom

Telephone
+1-978-897-6487
+1-877-SLEEPYCAT
(Toll-free, USA only)

wp_perf_0705c