Sorin Stancu-Mara, Peter Baumann, Vladislav Marinov

A Comparative Benchmark of Large Objects in Relational Databases

Technical Report No. 14
January 2008

School of Engineering and Science

A Comparative Benchmark of Large Objects in Relational Databases

Sorin Stancu-Mara, Peter Baumann, Vladislav Marinov
School of Engineering and Science
Jacobs University Bremen gGmbH
Campus Ring 1
28759 Bremen
Germany
E-Mail: s.stancumara@jacobs-university.de, p.baumann@jacobs-university.de, v.marinov@jacobs-university.de
http://www.jacobs-university.de/

Summary

Originally Binary Large Objects (BLOBs) in databases were conceived as a means to capture any large data (whatever large meant at the time of writing) which, for whatever reason, cannot or should not be modeled relationally. Today we find images, movies, XML, formatted documents, and many more data types stored in database BLOBs. A particular challenge obviously is moving such large units of data as fast as possible, hence performance benchmarks are of interest.

However, while extensive evaluations have been undertaken for a variety of SQL workloads, BLOBs have not been the target of thorough benchmarking up to now. TPC and SPC-2 standards do not address BLOB benchmarking either.

We present a comparative BLOB benchmark of the leading commercial and open-source systems available under Unix/Linux. Commercial DBMSs are anonymised, open-source DBMSs benchmarked are PostgreSQL and MySQL.

Measurements show large differences between the systems under test, depending on various parameters. A surprising result is that overall the open-source DBMSs in most situations outperform commercial systems if configured wisely.

ACM Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems – *query processing, relational databases.* D.2.8 [**Software Engineering**]: Metrics – *Performance measures.*

General Terms
Measurement, Performance.

Keywords
BLOB, comparative benchmark.

Contents

1 Motivation 3

2 Benchmark Setup 4
2.1 System Setup 4
2.2 Query Workload 4
2.3 Common Configuration Parameters 5

3 Systems Under Test 5
3.1 SystemA 6
3.2 SystemB 6
3.3 PostgreSQL 7
3.4 MySQL 7

4 Results 8
4.1 SystemA 8
4.2 SystemB 9
4.3 PostgreSQL 11
4.4 MySQL 13

5 Comparison and Discussion 14
5.1 Write Performance 14
5.2 Read Performance 14
5.3 Outliers 16

6 Conclusion 16

References 18

1 Motivation

Binary Large Objects (BLOBs) form a practically very relevant data type in relational databases. Introduced by Lorie in the eighties [5] as the equivalent of operating system files, that is: variable length byte strings without any further semantics known to the DBMS, they are used to hold a variety data that database designers cannot – or do not want to – map to tabular structures. Examples include text documents, images, and movies. Frequently BLOBs are relatively large against the page size of a database. Database implementers, therefore, have raised the size limit from 32 kB to 2 GB successively, sometimes even up to 128 TB, to accommodate text, XML, formatted documents, images, audio, and video data.

One use of BLOBs is for database management of large multi-dimensional arrays [1], for example 2-D satellite maps, 3-D x/y/t satellite image time series or x/y/z geophysics data, and 4-D x/y/z/t climate and ocean data. Array databases internally partition arrays into subarrays (tiles) where each tile goes into one database BLOB, forming the unit of disk access. Tiling strategies allow tiles of different size and shape, effectively providing a tuning parameter to the database designer.

Researching on best practices for tile tuning in array databases is one motivation for benchmarking BLOBs. Another interest grounds on the observation that running the same array DBMS on the same platform with the same application and identical data sets (earth observation imagery, in this case) conveys remarkable differences in performance on different relational DBMSs.

It turned out that comparative database benchmarks on BLOB behavior are not available. The Transaction Processing Council has defined several benchmark suites to characterize online transaction processing workloads and decision support workloads [1], however, none of the benchmarks developed so far capture the task of managing large objects. SPC-2 [8] benchmarks storage system applications that read and write large files in place, execute large read-only database queries, or provide read-only on-demand access to video files, yet SPC-2 does not consider database BLOBs.

Some papers and technical reports have been published which address the performance of large objects in relational databases. In [7] authors compare DBMS and file system by addressing fragmentation; they conclude that BLOBs of less than 256KB should be stored in a database while BLOBs over about one MB are more efficiently handled by a file system; in the grey zone inbetween application considerations prevail. This benchmark, however, focuses exclusively on Windows/NTFS and a Windows-only DBMS, while our interest is in Unix/Linux environments. Further, there is a number of white papers by database vendors detailing BLOB handling in their particular framework [4][3][2].

This lack of publicly available coherent benchmark results motivated us to conduct our own comparative performance analysis of BLOB management across different DBMSs, incorporating both open-source and commercial systems. The results of this investigation we present in this contribution.

The remainder is organised as follows. In the next section we detail the benchmark setup. In Section 3 the systems under test are described. Section 4 presents the measurements obtained for each DBMS. We compare and discuss these in Section 5, and finally draw conclusions and give an outlook in Section 6.

2 Benchmark Setup

In this Section we list the overarching parameters of the benchmark run; the DBMS specific setups are presented in Section 3.

2.1 System Setup

All DBMSs were installed on a 3.0 GHz Pentium 4 PC (hyperthreading enabled) with 512 MB main memory and 7,200 rpm IDE disks. The operating system used was a vanilla SuSE Linux 9.1 (i586). All demons, except sshd, were stopped during the experiments.

Database files were allocated on a different disk than the operating system files to avoid any interference in disk access, e.g., through intermediate access to /tmp/ or /var.

To further decouple disk access, the operating system disk was connected via the primary IDE channel, while the disk holding the databases was connected via the secondary IDE channel.

2.2 Query Workload

The database schema used is straightforward: a table *LargeObjects* with an integer primary key attribute, *id*, and a BLOB attribute, *largeData.*

We measured reading and writing of BLOBs, always writing BLOBs as whole which in our opinion is the by far prevailing type of access.

Sizes of the large objects were arranged logarithmically as 1K, 2K, 5K, 10K, 50K, 100K, 500K, 1M, 5M, 10M. An additional out-of-pattern value of 3,999 bytes considers the inlining facility for BLOB size below 4k supported by SystemA. For each BLOB size ten different BLOBs were generated using /dev/random for filling them. Using different contents (i.e., not copying by re-using BLOBs already sent to the server) made sure there is no way of caching. The resulting times were averaged over the ten BLOBs.

Database access was programmed in C/C++, making use of best practices for each DBMS. As BLOB access is highly unstandardized, each system follows its own pattern for BLOB access, ranging from a single SQL statement to a sequence of statements to C API calls. The benchmarking clients, therefore individually use embedded SQL or the API provided.

On an abstract SQL level, the write queries would follow this schema:

```
INSERT INTO LargeObjects (id,largeData)
VALUES (:id,:largeDataBuffer)
```

Read queries adhere to this pattern:

```
SELECT largeData
INTO   :largeDataBuffer
FROM   LargeObjects
WHERE  id=…
```

All measurements of BLOB access were strictly confined to the DBMS call; establishing the BLOB data in client main memory was not in the timing scope, and no file writing of BLOBs read was done.

The complete benchmark suite – including source code, compile documentation, and consolidation scripts – will be made available on the ACM SIGMOD benchmark web page.

2.3 Common Configuration Parameters

Installation of all DBMSs was performed following standard recommendations in the setup procedures. Tuning parameters were handled in three different ways.

Basic parameters were set uniformly to achieve comparability; this involved:

- Database page size (which usually is preset during database installation, and then fixed). For the sake of comparability, page size was uniformly set to 4K, as one of the systems, PostgreSQL, only supports this page size.
- Logging, which plays a central role due to the expensive generation of undo and redo log entries for large objects. In our tests logging uniformly was disabled.
- Preparing queries was not used; brief tests showed that in the situation on hand prepared queries do not contribute measurably to overall execution times.
- MySQL, SystemA, and SystemB used prepared statements; it has been made sure that only binding and execution was timed. PostgreSQL relied on a C API without queries.

Secondly, if a system offers some particular speedup option we activated it to allow each system to perform optimally. Two such options we found relevant:

- Usage of separate buffers for handling BLOBs vs. usage of the same buffer for handling all database data. Whenever supported we established dedicated BLOB buffers.
- Inline storage of BLOBs, i.e, storing them inside the rows of the normal table as opposed to factoring out BLOB data into separate tables and table spaces. Inlining was enabled whenever supported.

The third category, finally, comprises parameters subject to benchmarking, including BLOB buffer page size. We tested performance with various page sizes.

3 Systems Under Test

Our choice of DBMSs for benchmarking was determined by the DBMSs market dominance, by their availability under the Unix/ Linux platform, and finally we wanted to include both commercial and open-source DBMSs. To keep with vendor restrictions we anonymise the two commercial DBMSs to SystemA and SystemB, resp. The open-source DBMSs benchmarked are PostgreSQL version 8.2.3 and MySQL version 5.0.45 (which are the currently recommended stable versions).

3.1 SystemA

Overview

SystemA, a commercial DBMS, provides support for defining and manipulating large objects of up to 2GB. Further, there is the LONG datatype; its usage is not recommended, though. There are two alternatives for storing LOB data in an SystemA database. If the size of the LOB is less than 3,964 bytes (sometimes also reported to be 4,000 bytes) then it can be stored in-line with the other column data. Otherwise, LOBs are stored out-of-line in blocks of certain size in a segment of type LOBSEGMENT; a 20-byte LOB locator is stored in the base table as placeholder. Whenever a table containing a LOB column which is to be stored out-of-line is created two segments are created to hold the specified LOB column. These segments are of type LOBSEGMENT and LOBINDEX. The LOBINDEX segment is used to access LOB chunks that are stored in the LOBSEGMENT segment. The LOBINDEX and the LOBSEGMENT are stored in the same table space as the base table containing the LOB. Whenever a LOB object, which is stored in-line, is updated and its size grows beyond the abovementioned limit then it is automatically moved to out-of-line storage. Whenever LOB data is stored in-line it is manipulated directly via the buffer cache. Otherwise, the LOB locator is used to lookup the LOBINDEX which stores the range of the addresses of the LOB data in the LOBSEGMENT.

LOB objects in SystemA can be stored and retrieved using LOB locators, but it is not possible to manipulate them via functions and operators. A further restriction is that only one LOB column per base table is allowed.

Benchmark Configuration

During database server setup we specified that the database files should be managed by the operating system file system instead of by the automatic storage management center of the database server. This would give us better control w.r.t. details concerning LOB I/O operations during the experiments.

An attempt was made to modify memory parameters in the System Global Area (SGA) and the Program Global Area (PGA). Changing the value of the buffer cache, however, made the database server crash. Thus, we decided on using the default value of 64MB for the buffer cache and deactivate reading/writing via the buffer cache using other tools during the experiments. Consistent reading was disabled for better throughput. The buffer cache was disabled. The CHUNK parameter (which determines LOBSEGMENT allocation size) turned out to be influential; to capture this we benchmarked different CHUNK sizes.

3.2 SystemB

Overview

SystemB, the second commercial DBMS, stores LOBs in a separate (auxiliary) table space which has a completely different format from the base table space. LOB entries can occupy up to 2GB. Each LOB value stored in the LOB table space is associated with a LOB map page with descriptive information about the value as well as the addresses of the chunks where the actual LOB value is stored in the auxiliary table space.

In order to avoid materialization of LOBs as much as possible SystemB makes use of LOB Locators. Unlike a SystemA LOB locator, which is simply a reference to the LOB value's storage address, a LOB locator in SystemB is a host variable which can contain a value that represent a single LOB instance. SystemB has support for parallel I/O and integrated LOB function optimizer which can reduce I/O overhead.

Benchmark Configuration
Installation followed the recommended standard procedure, no particular tuning was performed. BLOB access was performed using the C API, as SystemA claims to be faster via its native API than via embedded SQL.

3.3 PostgreSQL

Overview
Open-source PostgreSQL offers two variants for large objects: its large object facility and the TOAST data type [7]. Size of a TOAST object is restricted to 1 GB, large objects can span up to 2 GB.

PostgreSQL uses a fixed page size of 4 K and does not allow a row to span multiple pages. Therefore, in both techniques the LOB data are partitioned internally before being stored in a table.

Large objects are placed in a single system table called *pg_largeobject*. A large object is identified by an OID assigned when it is created. Each large object is broken into segments or "pages" small enough to be conveniently stored as rows in the *pg_largeobject* table. The table contains three attributes: a LOB identifier, a page number, and the actual data. LOBs in the *pg_largeobject* table can be accessed via streams and the *libq* API library.

Benchmark Configuration
Tuning and configuration of the PostgreSQL database instance was performed by modifying the *postgresql.conf* configuration file, followed by recompilation of the server.

3.4 MySQL

Overview
Open-source MySQL provides three types of BLOBs: *tiny* (up to 256 bytes), *medium* (up to 64 K) and *long* (up to 4G) [6]. Besides their sizings MySQL doesn't offer any other parameter directly impacting blob performance.

However, the storage engine used to hold the table containing the BLOB can be selected and configured individually per relation via the create table command. Currently available engines include InnoDB (which provides transaction capabilities), MyISAM (the new ISAM engine, which is used by default), Archive, BerkeleyDB (which is deprecated and will be phased out), FEDERATED (which allows tables to be stored on remote machines), and MEMORY (which stores data in the system heap but doesn't allow BLOBs). BLOB data uniformly are kept in the same file as the table containing the BLOB attribute.

In summary, engines most useful for BLOB management are InnoDB, MyISAM, and Archive, and these are the ones benchmarked.

Benchmark Configuration
Installation followed the recommended standard procedure, no particular tuning was performed. BLOBs were inserted and retrieved as byte arrays through the C API.

4 Results

4.1 SystemA

Write performance. This was measured for CHUNK sizes of 8K and 32K. Results for BLOB sizes up to 50k are presented in Figure 1, those for 50K up to 10M in Figure 2).

Obviously storing small BLOBs inline speeds up write performance. Further, writing small BLOBs (less than 24K) takes almost constant time independent of the BLOB size, and values are very close for CHUNK size 8K and 32K. This can be explained by the CHUNK granularity: Even for a 1K BLOB a complete CHUNK has to be written to the database since this is the smallest SystemA storage unit. Hence, if storage waste by underfull CHUNKs is not an issue then the number of CHUNKs transported should be minized by setting CHUNK size to expected BLOB size; SystemA has a limit, however, on 32K.

Read Performance. The BLOBs thus written were then measured for retrieval performance, using a separate program run.

Reading small BLOBs (below 50K, see Figure 3) is not affected by the CHUNK size. Similarly to the write performance discussion the reason is that only a small number of CHUNKs has to be read. We note that inline storage of small BLOBs is a bit more beneficial for the read performance, although the difference measured is just 1ms. On the other end (Figure 4), a high CHUNK size has a clear influence on read times for BLOBs of 1M and above.

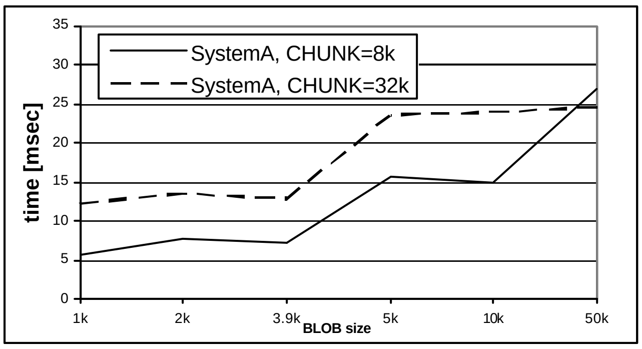

Figure 1: SystemA small BLOB write performance

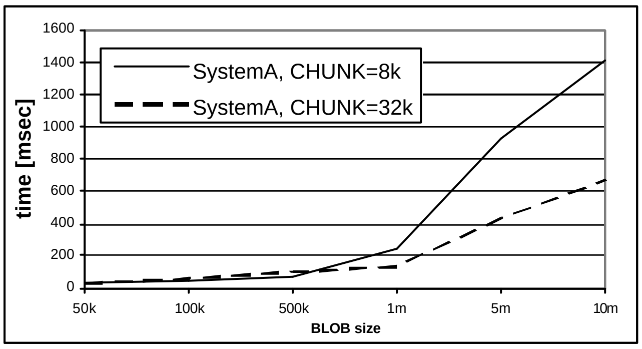

Figure 2: SystemA large BLOB write performance

The threshold where access time starts to correlate significantly with BLOB size is above 100K.

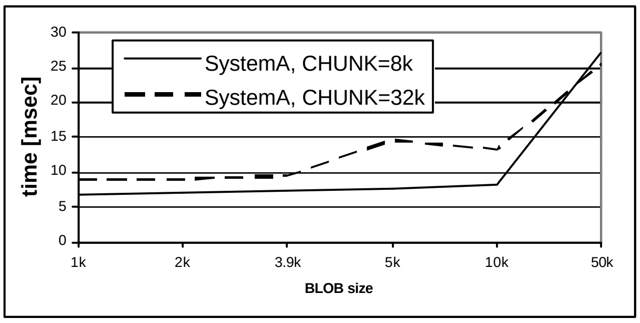

Figure 3: SystemA small BLOB read performance

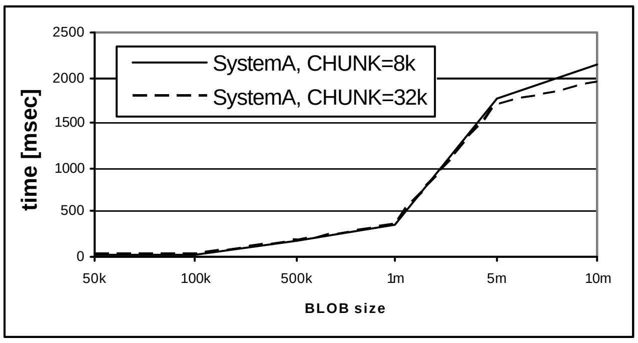

Figure 4: SystemA large BLOB write performance

4.2 SystemB

Measurements were conducted in four tablespaces of type *large* with page sizes of 4K, 8K, 16K, and 32K; with each tablespace a buffer pool with same page size was associated. The BLOB column size was set to the BLOB size during table creation in order to optimize performance. Modification of the number of prefetched pages did not affect the results, hence this parameter is omitted here.

Write performance. Although we performed the measurements for different values of the buffer pool and table space page size, in Figure 5 significant differences can only be observed between the 4k line against the other lines.

The local maximum at 5K is mainly determined by a peak of 167ms in otherwise an average of 17ms. This situation was reproducible, so we assume that a buffer flushing action occurs here. Generally speaking, write times below 50k were observed to be relatively divergent, hence hard to predict.

Increasing page size beyond 4K for large BLOBs (see Figure 6) improves performance, but not substantially. Thus, we conclude that SystemB's write performance is not significantly influenced by the buffer pool and the table space page size; that said, actually a small page size of 4K can be recommended.

The threshold where write times increase starts to correlate with the BLOB size is approximately 50K.

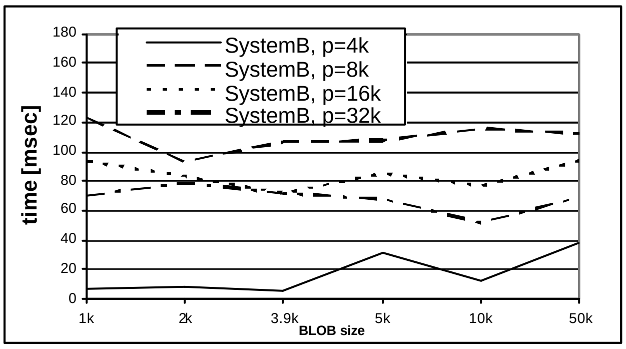

Figure 5: SystemB small BLOB write performance

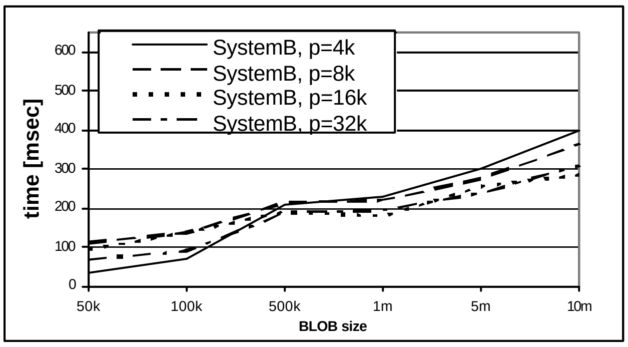

Figure 6: SystemB large BLOB write performance

Read Performance. As Figures 7 and 8 indicate, changing buffer pool and table space page size does not affect SystemB's read performance significantly. Interestingly, however, reading BLOBs of size 5M+ is faster for smaller page sizes.

The peak at 100K is due to spikes observed; see also the discussion in Section 5.

Still our conclusion is that the BLOB read performance of SystemB is not significantly affected by the page size of buffer pool and table space.

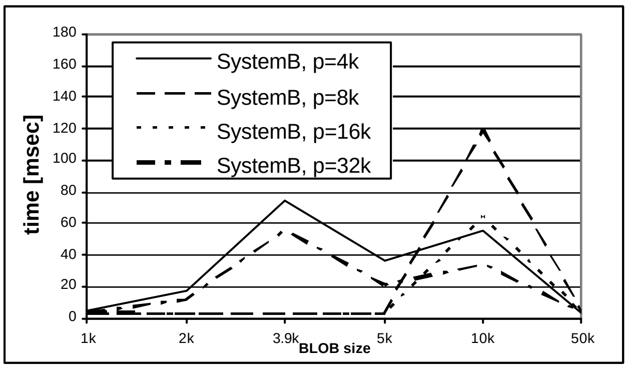

Figure 7: SystemB small BLOB read performance

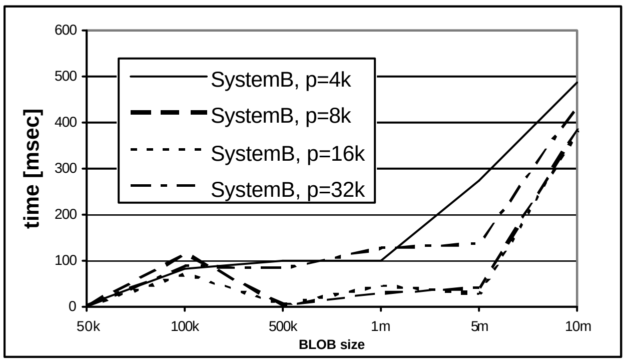

Figure 8: SystemB large BLOB read performance

4.3 PostgreSQL

Measurements were performed for several combinations of (BLCKSZ, LOBLKSIZE) values. An attempt was made to read and write BLOBs when LOBLKSIZE was set to the same value as BLCKSZ, however the database crashed with an error showing that the row size of the *pg_largeobject* table was too big and inconsistent with the BLCKSZ value. Thus in all measurements the LOBLKSIZE is either 1/4 or 1/2 of the BLCKSZ. B, l, and b denote BLCKSZ, LOBLKSIZE, and application buffer size, resp.

Write performance. We observe that write performance for small BLOBs (1K - 24K) remains almost unchanged for all values of BLCKSIZE and LOBLKSIZE.

The threshold at which write time starts to increase significantly with BLOB size is about 50K.

The homogeneous behavior extends into large BLOB sizes. The curve in Figure 10 might suggest that for larger BLOBs from 10M onwards b=4k tends to be preferable.

In summary, PostgreSQL BLOB performance is only marginally affected by the sizing parameters, and overall shows a very regular behavior.

Read Performance. Just as before we observe that small BLOB access is almost constant (Figure 11), hence independent from BLOCKSZ and LOBLKSIZE.

The threshold where we observe that read time tends to depend on the BLOB size is about 50-100K.

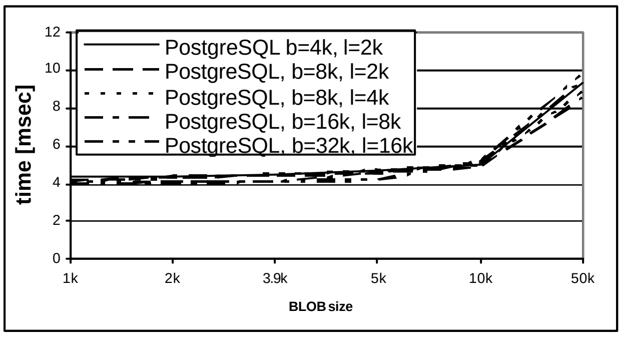

Figure 9: PostgreSQL small BLOB write performance

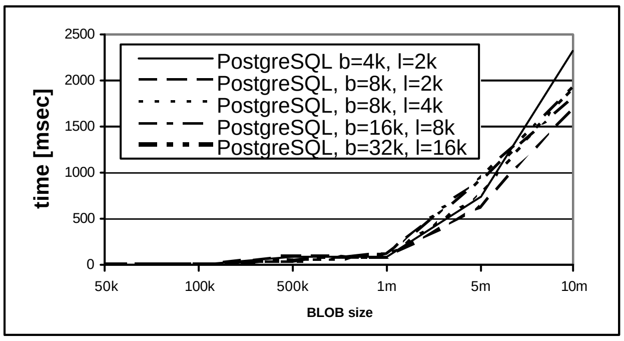

Figure 10: PostgreSQL large BLOB write performance

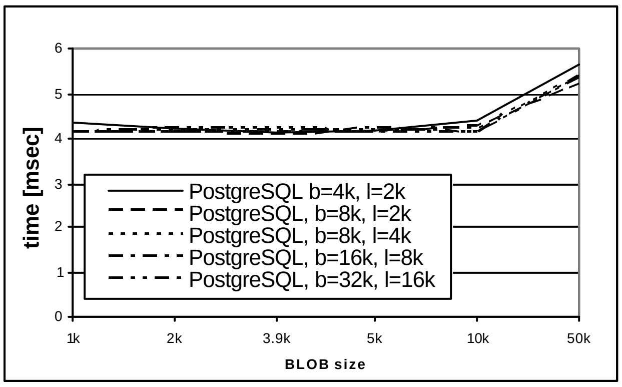

Figure 11: PostgreSQL small BLOB read performance

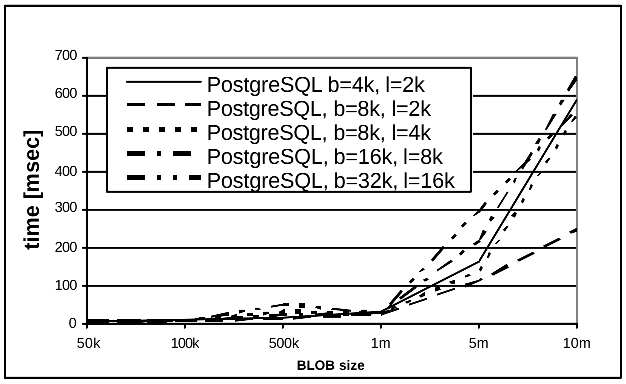

Figure 12: PostgreSQL large BLOB read performance

4.4 MySQL

MySQL was benchmarked with the three different engines, as discussed in Section 3.4.

Write performance. While ARCHIVE and MyISAM show an almost linear, extremely fast write behavior, the huge InnoDB value changes constitute a prime eyecatcher. Basically, looking into the individual measurement values, InnoDB shows the same good behavior as the other engines. However, very frequent large spikes occur with random times and heights. For example, the very first 1K write access takes 675ms, as opposed to the second and subsequent ones with 0.27 ms. Further spikes follow closely, but irregularly.

From the documentation it appears that this is related to the engine's log file writing. The initial spike might be due to some initialization overhead of the InnoDB engine. Notably the engine was cold started for the measurements – like with every system we measured – so that previous activity could not contribute.

Large BLOB sizes, on the other hand, behave as expected. InnoDB and MyISAM show a practically identical curve while ARCHIVE performs substantially better the larger the BLOBs get.

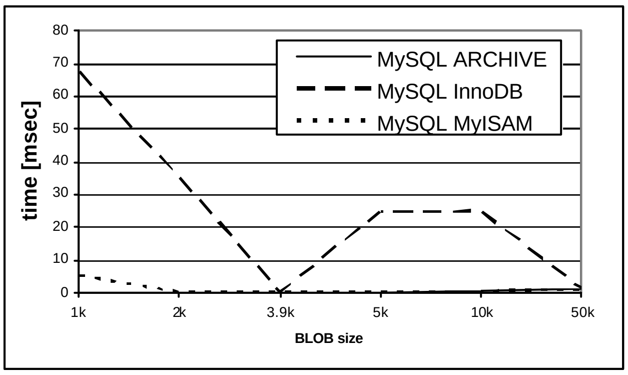

Figure 13: MySQL small BLOB write performance

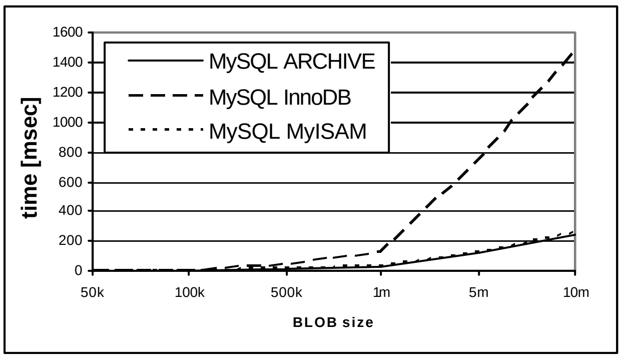

Figure 14: MySQL large BLOB read performance

Read Performance. As opposed to write access, read times appear very regular. Again, ARCHIVE performs best, particularly for large BLOBs.

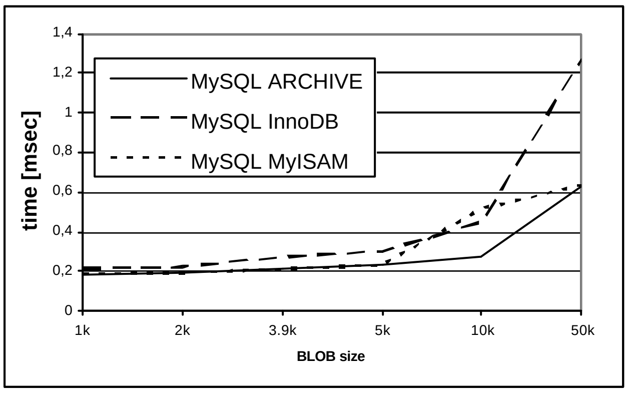

Figure 15: MySQL small BLOB read performance

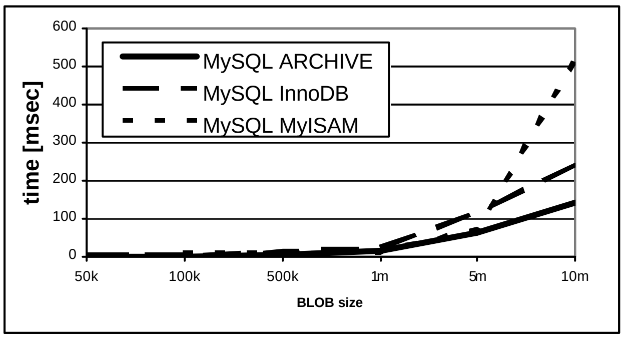

Figure 16: MySQL large BLOB read performance

5 Comparison and Discussion

Based on the previous results we now can compare the results; for each DBMS, the best setup found is used below. This means that for different situations different settings per DBMS have been used – for example, with low BLOB sizes SystemA is best with CHUNK=8K whereas for large BLOBs it performs best with CHUNK=32K.

5.1 Write Performance

Figures 18 and 19 list the optimal write access times for each DBMS. We observe that MySQL has by far the best results, and very stable over the complete range. PostgreSQL shows a similar behavior, but substantially slower. Interestingly the inlining facility of SystemA does not get it even close to MySQL when faced with small BLOBs. SystemA is slower than PostgreSQL, with a relative direct correlation between write time and tile size. SystemB faces spikes which deteriorate the overall result, otherwise it is about comparable with SystemA.

For large BLOBs the anomalies vanish. Ranking is about the same, with the exception of PostgreSQL which rapidly loses performance beyond about 2M.

5.2 Read Performance

The optimal read performance results for each system are shown in Figures 20 and 21. Small BLOBs show an almost constant timing, the 10K outlier for SystemB is due to one of the spikes discussed earlier. The overall ranking is as before: MySQL followed by PostgreSQL, then SystemB (except for the spikes), finally SystemA.

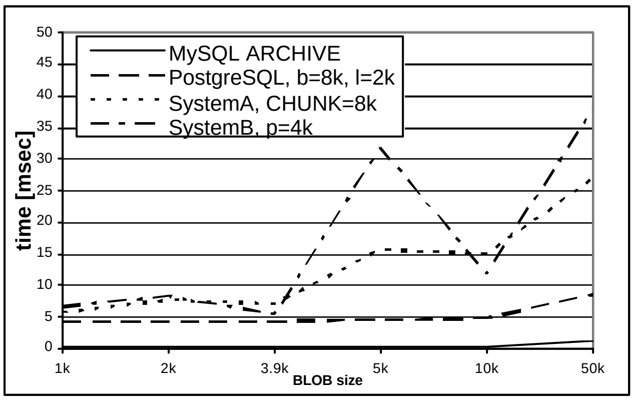

Figure 18: small BLOB write performance comparison

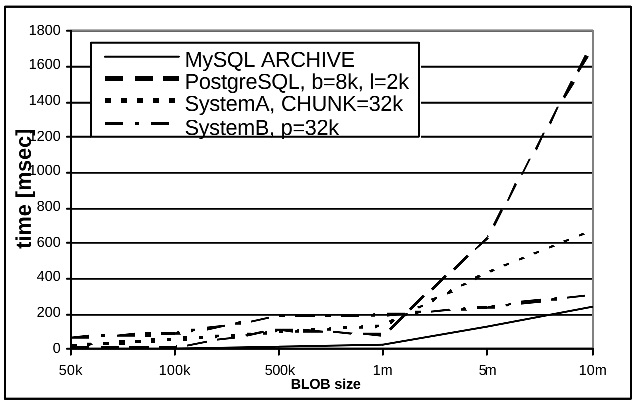

Figure 19: large BLOB write performance comparison

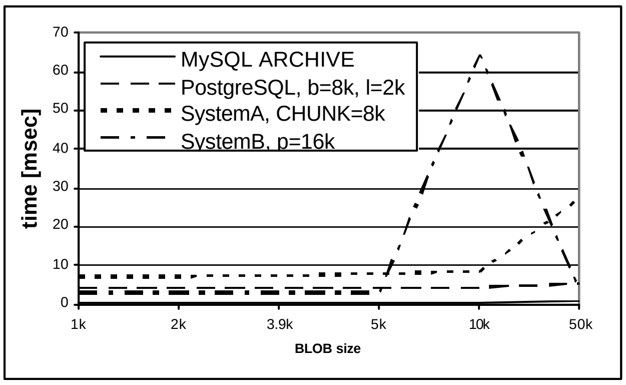

Figure 20: small BLOB read performance comparison

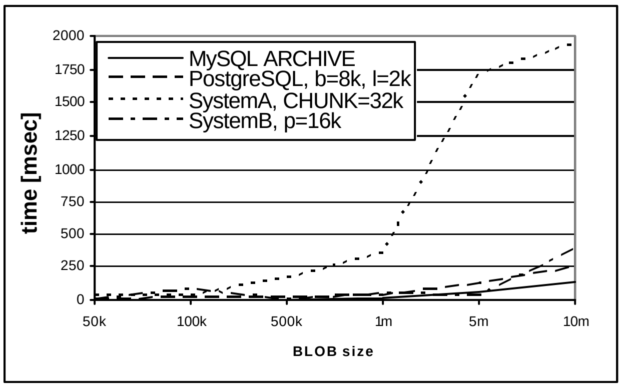

Figure 21: large BLOB read performance comparison

For large BLOBs, both MySQL and PostgreSQL demonstrate excellent scalability; SystemB does so too, up to a size of about 5M where performance starts to deteriorate. For SystemA, this point is already reached at a few 100K.

5.3 Outliers

A general phenomenon observed was – not surprisingly – that severe spikes influence the average values. This obviously is due to internal flushing of main memory buffers. A typical example is shown in Table 1.

Table 1: sample measurement results for SystemB, p=4K, BLOB size written 2K [msec]

4,176	4,048	4,913	4,122	3,105	4,752	3,207	3,922	4,1	138,58

Table 2: sample measurement results for MySQL, InnoDB engine, BLOB size written 1K [msec]

675	0,268	0,251	0,26	0,293	0,262	0,265	0,265	0,269	0,261

An extreme case we observed with MySQL (Table 2) where a spike with 675ms occurred in a neighborhood of 0.26ms averages.

As this phenomenon will happen in normal operation as well, spikes have not been masked and contribute to the averaging over the multiple measurements.

In summary, SystemA and PostgreSQL show an overall homogeneous behavior. SystemB and MySQL tend to have occasional delays which can exceed normal average processing time by two orders of magnitude.

6 Conclusion

By performing a comparative BLOB benchmark we tried to fill a gap which is relevant for many of today's applications using multimedia, scientific, semistructured, or similar data.

The comparative benchmark conducted reveals remarkable differences in BLOB performance. The great surprise was that open-source DBMSs outperform commercial DBMSs in most situations, with MySQL being the clear champion. This most likely is not due to a particular optimization in MySQL and PostgreSQL – as somebody has put it, the more features a DBMS has the more these can interfere. Whatever the reason is, this opens up vistas for cost-efficient high-volume service installations.

Generally speaking, BLOB performance started to depend only on BLOB size once the basic tuning and sizing parameters are exceeded sufficiently, that is: beyond 50k. Performance of BLOBs seems to deteriorate above a few MB.

This leads to the inverse question: given some data volume to be stored and retrieved, which system and which BLOB size is optimal?

Based on our measurements an estimation is provided in Figure 22 showing extrapolated access time for reading 1GB of data; note the logarithmic time scale. Obviously this has to be interpreted very carefully and needs further experimental underpinning.

It appears that every system has its individual optimum. For the champion, MySQL, there is a weak minimum at a BLOB size of about 100K to 1M. Both PostgreSQL and SystemB seem to have its best performance with somewhat larger BLOBs, between 1M and 10M. No clear optimum is visible for SystemA; values of 1M+ seem appropriate, though. Notably performance varies by two orders of magnitude between 100K / MySQL and 10K / SystemB.

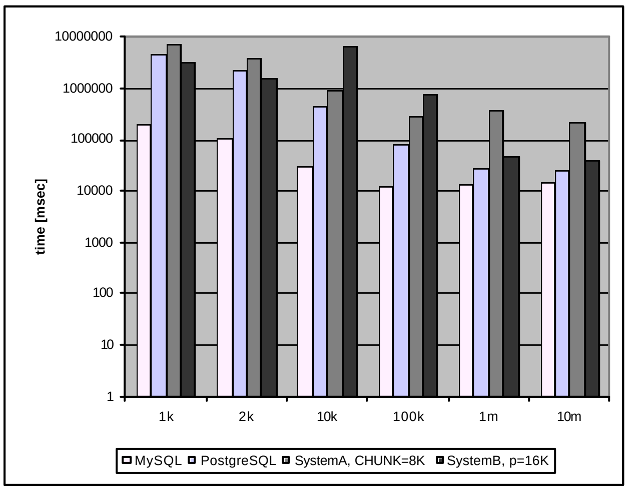

Figure 22: time needed for reading 1GB of BLOBs

Performance isn't everything. Even for heavily BLOB-geared applications BLOB performance is but one DBMS selection criterion. Large-scale applications will value stability and further service parameters. Concerning stability, both a commercial and an open-source DBMS crashed, fortunately during installation and not in operation. We observe, however, that SystemA, SystemB, and PostgreSQL are running stable for several years now under industrial conditions in conjunction with rasdaman holding geo imagery BLOBs in excess of 1 TB, in case of PostgreSQL approximately 13 TB. For MySQL we have no experience values as it has not been used in our geo service application up to now.

With our benchmark we want to share our experience with the community so as to allow incorporating it into the DBMS selection process, and also give hints to DBMS implementors on large object optimization potential.

As we have observed that in most cases documentation does not provide information on the best choice for good BLOB performance we aim at contributing to best practices for BLOB tuning. Measurements conducted and results obtained suggest the following recommendations for best BLOB performance of the DBMS:

- Logging should be turned off whenever feasible to avoid the expensive generation of undo and redo log entries for large objects.
- A dedicated buffer pool and a dedicated table space for handling and storing BLOB data is advantageous; from the systems benchmarked only SystemB supports this option.
- Choice of both database page size, buffer pools size, and BLOB structures on disk should be dictated by the size of the BLOBs to be handled. For small BLOBs, smaller page size avoid wasting disk space. For large BLOBs, a large page size is helpful, which usually is limited to 32K. As these parameters often are fixed during database creation some upfront reflexion on the expected BLOB sizes is feasible.

One further determining factor is storage management, in particular: clustering of BLOB chunks. We repeated BLOB read and write in different orders; however no significant difference was observed. We intend to focus on this issue in future, though, as it is of particular importance for our work on array databases where BLOBs containing partitions of multidimensional arrays should be clustered spatially. Ultimately, we would like to control BLOB placement on disk, possibly via some hinting interface where the application can inform the DBMS upfront about the intended BLOB access patterns.

References

[1] Baumann, P.: Large-Scale Raster Services: A Case for Databases. Invited keynote, 3rd Intl Workshop on Conceptual Modeling for Geographic Information Systems (CoMoGIS), Tucson, USA, 6 - 9 November 2006. In: John Roddick et al (eds): Advances in Conceptual Modeling - Theory and Practice, 2006, pp. 75 - 84.

[2] Bruni, P., Becker, P., Dewert, M., Riehle, B., Large Objects with DB2 for z/OS and OS/390, *IBM Redbook*, June 2002.

[3] Jegraj, V., LOB Performance Guidelines, *Oracle White Paper*, 2006.

[4] Lehman, T., Gainer, P., DB2 LOBs: The Teenage Years, *Proc. ICDE 1996*, IEEE Computer Society, Washington/USA, pp. 192-199.

[5] Lorie, R.A., Issues in databases for design transactions, in: Encarnaçao, J., Krause, F.L. (eds.), *File Structures and Databases for CAD*, North-Holland Publishing, IFIP, 1982.

[6] n.n., MySQL documentation, dev.mysql.com/doc

[7] n.n., Postgresql documentation, www.postgresql.org/docs/

[8] n.n., SPC Benchmark-2 (SPC-2) Official Specification Version 1.0, Storage Performance Council, www.storageperformance.org/specs/

[9] n.n., Transaction Processing Council, www.tpc.org

[10] Sears, R., van Ingen, C., Gray, J., To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? *Microsoft Research Technical Report*, University of California at Berkeley, April, 2006.

[11] Shapiro, M., Miller, E., Managing databases with binary large objects, *16th IEEE Symposium on Mass Storage Systems*, 1999, pp. 185-193.