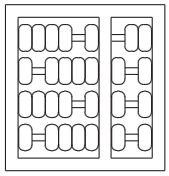


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Evaluation of a Read-Optimized Database for Dynamic Web Applications

A. Supriano, G.M.D. Vieira, L.E. Buzato

Technical Report - IC-07-14 - Relatório Técnico

May - 2007 - Maio

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

Evaluation of a Read-Optimized Database for Dynamic Web Applications

Anderson Supriano Gustavo M. D. Vieira* Luiz E. Buzato

Instituto de Computação — Unicamp
Caixa Postal 6176
13083-970 Campinas, São Paulo, Brasil
anderson@supriano.com, {gdvieira, buzato}@ic.unicamp.br

Abstract

In this paper we investigate the use of a specialized data warehousing database management system as a data back-end for web applications and assess the performance of this solution. We have used the Monet database as a drop-in replacement for traditional databases, and performed benchmarks comparing its performance to the performance of two of the most commonly used databases for web applications: MySQL and PostgreSQL. Our main contribution is to show for the first time how a read-optimized database performs in comparison to established general purpose database management systems for the domain of web applications. Monet's performance in relation to MySQL and PostgresSQL allows us to affirm that the widely accepted assumption that relational database management systems are fit for all applications can be reconsidered in the case of dynamic web applications.

1 Introduction

Relational database management systems (RDBMSs) are without any doubt one of the most successful products of computer science applied research. Since the early 1980s, research prototypes were turned into commercial products, these products were adopted by many clients and now run most of the world business processes. The acronym "RDBMS" is considered by most application developers and businesses executives an unquestionable synonym for storage, processing, and retrieval of data and,

*Financially supported by CNPq, under grant 142638/2005-6.

more generally, for the information technology field as a whole. But, as companies and technologies evolve, the traditional client-server software architectures are being replaced by multi-tier software architectures and applications fit for the web. Businesses have become connected to each other and to their clients, by interconnecting their systems via web services. In this new scenario, on-line transaction processing (OLTP) RDBMSs, continue to be important, but a significant portion of the companies processes acquire a new profile. Web applications for reporting business sales, marketing, management reporting, business process management (BPM), budgeting and forecasting, financial reporting, mapping, e-commerce, etc, become key to the continuous operation of businesses. Despite this diversity of applications, a bird's eye view of the software run by modern companies is certainly going to show RDBMSs behind not only standard OLTP applications but also as engines of on-line analytical transaction processing (OLAP) and web applications.

This technological uniformity has its price. The database market is now ruled by a handful of vendors such as Microsoft, Oracle, IBM and Sybase, that pioneered the "One size fits all" philosophy for their products. They maintain and sell only a single code base to all their clients, simplifying the development and marketing of their DBMSs. This approach is very successful not only for vendors but also for clients, who now expect to find a familiar programming interface in every DBMS. The downside of this practice is that, by trying to serve as many clients as possible with the same product, database companies may be shipping applications based upon OLTP optimized databases where in fact other database models and solutions may be more appropriate. In a recent paper, Stonebraker and Çetintemel [11] argue that there are domains where a 10 fold increase of performance is expected if the database is optimized or, more radically, if the RDBMS is completely replaced by a different approach to data management. Specifically, they list the domain of data warehousing and OLAP as examples of the former and stream processing as an example of the later. Varying application requirements and evolution of hardware architectures are the main reasons for this 10 fold increase, and this order of magnitude increase in performance is enough to justify the pursuit of these alternative solutions.

We believe that the construction of dynamic web applications is a domain with a potential for order of magnitude performance increase. Even though web applications are generally considered to be in the OLTP domain, in general these applications deviate from the usual write-intensive pattern usually associated with OLTP. Much of the dynamic content generated for the web is actually read-only, ranging from 85% read-only transactions in a web shop to nearly 99% read-only transactions in a bulletin-board application with a large user base [1]. Databases optimized for OLAP are read-optimized, thus their use could offer a simple way to increase the performance of web applications.

In this paper we investigate the use of a specialized data warehousing database management system as a data back-end for dynamic web applications and assess the

performance of this solution. We have used the Monet[1] [2] database as a drop-in replacement for traditional "One size fits all" databases, and performed benchmarks comparing its performance to the performance of two of the most commonly used databases for web applications: MySQL[2] and PostgreSQL[3]. The web application benchmark used was the industry standard TPC-W [13], representing a typical web store. Our main contribution is to show for the first time how a read-optimized database performs in comparison to established general purpose DBMSs for the domain of web applications. As more systems are built to take advantage of the 10 fold expected performance increases for some domains, we believe this type of cross domain comparison will became a useful tool for the definition of useful utility relations among different database managers and application domains.

The paper is structured as follows. Section 2 discusses possible matches and mismatches of read-optimized databases to the data storage requirements of dynamic web applications. Section 3 describes TPC-W and argues that it is the correct choice of workload to fairly compare Monet, MySQL and PostgreSQL. Section 4 describes the experimental setup used, and Section 5 is devoted to the analysis and discussion of the results. A summary of the contributions and final comments are contained in Section 6.

2 Read-Optimized Databases for Dynamic Web Applications

In the sixties, the concept of packet-switching enabled the creation of the Internet as we know today. The Internet model is very simple, routers inside the network forward data packets from a source computer to a destination computer, and application programs execute on the hosts connected to the edge of this network. The key protocols that enable network and end-to-end communication are collectively known as TCP/IP. This inherent simplicity, combined with a hierarchical naming application, the Domain Naming System, has fostered in the last few decades a proliferation of applications. Initially, during the eighties, these applications were mainly text-based: e-mail, remote access to computers, file transfer, newsgroups, etc. In the nineties, the evolution of personal computing with their graphical interfaces and faster, better LANs and WANs allows a second leap to happen: the World Wide Web. A simple HTML-enabled file transfer utility, the web browser, became the killer client for access to remote resources and applications, enabling easy *on-demand production* and display of content stored in corporate databases, giving birth to *web applications*.

[1] http://monetdb.cwi.nl/
[2] http://www.mysql.com/
[3] http://www.postgresql.org/

Initially created to implement simple, static, query interfaces to databases, web applications are now capable of computing their output dynamically as a function of the clients needs, querying and updating an underlying database as necessary. Today, these applications enable all types of activities and provide services as diverse as on-line shopping, home banking, auctions, airline reservation, etc. In most cases, web applications retain a property of their initial incarnation as database query interfaces: the majority of accesses to the database is read-only. Current estimates put the read/update ratio at 85% in a typical web shop and nearly 99% in a bulletin-board application with a large user base [1]. This read-intensive behavior is more related to OLAP workloads than OLTP workloads. However, until very recently, no one had questioned the use of conventionally optimized DBMSs to deploy dynamic web applications.

Data warehousing is a booming business, already representing 1/3 of the database market in 2005 [10]. The basic idea is to aggregate information from the many disjoint production systems in a single large database, to be used for business intelligence purposes. During the data aggregation process, data is copied from write-intensive OLTP systems to computation and read-intensive OLAP systems, while it is filtered, re-organized and indexed for analytical processing. A database optimized for this type of workload is different from a normal database as it is read-optimized. OLTP optimized databases store tables clustered by rows, because this allows a simple mapping of tables to a storage model optimized for the access of complete rows. Read-optimized databases, by contrast, typically need to access many rows of which only a few column values are used. Thus, read-optimized databases can be organized using vertical fragmentation, keeping data from one or more columns together [2, pp. 41-42]. Data organized in columns, among others optimizations, define a read-optimized database and is the central focus of the two more prominent new generation OLAP databases: Monet [2, 3] and C-Store [12].

Monet has interesting features, it was designed to extract maximum database performance from modern hardware, especially for complex queries. It optimizes the use of processor registers and cache memory through data structures optimized for main memory, vertical fragmentation and vectorized query execution. Monet is being actively developed as an open source project, has a SQL query interface and bindings to many languages such as Java, Perl and Ruby. It is a mature project that closely matches the expected functionality of a DBMS. C-Store's main focus is efficient *ad-hoc* read-only queries and the main optimizations used are vertical fragmentation, careful coding and packaging of objects, implementation of non-traditional transactions and use of bitmap indexes. Despite all these interesting characteristics, C-Store is currently only a prototype with restricted availability.

To assess the applicability of a read-optimized database to the construction of web applications, we decided to test one of these systems as a drop-in replacement for a traditional DBMS in a web application built using the Java language and the

Java Enterprise Edition (JEE) stack. This is a very common application development platform and both MySQL and PostgresSQL have native bindings to Java through the industry standard JDBC interface. We selected Monet due to its maturity and the fact that it also has a native binding to Java through JDBC. We were not able to include C-Store in the experiment due to its prototypical stage of development and availability.

2.1 Monet

A detailed description of Monet can be found in P. A. Boncz's thesis [2]. In this section, we summarize the features of Monet's software architecture and functionality that are relevant to our experiment.

Use of binary tables: Monet takes the vertical fragmentation of relations to an extreme and only uses binary association tables (BATs). These are tables with only two columns: one representing a object identification number (OID) and the other the actual object data. A row in a traditional relation is obtained by the join of the contents of several of these binary tables with the same OID. Figure 1 shows how Monet fragments a relation into BATs. Actually, OIDs are completely under the control of Monet and can be made sequential, thus internally each binary table is implemented by a single array and elements are retrieved by direct array indexing.

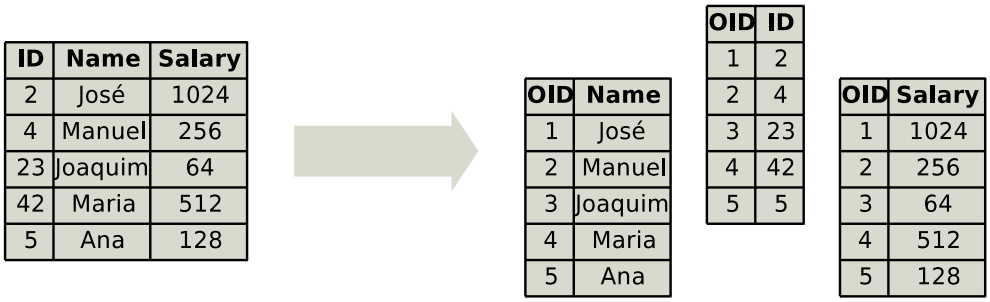


Figure 1: From a relation to BATs.

Optimized for main-memory execution: Large memories available to current systems are used by Monet to improve its performance. This happens in several fronts, from basic data structures to query processing. The main data structure is the BAT, *completely* loaded in main memory and indexed using hashes and balanced search trees. Query processing is not concerned with optimizing disk I/O, as all data is already in memory, but it tries to optimize memory I/O and the utilization of cache memory and processor registers.

Single back-end for multiple front-ends: Monet handles the query language parsing, query rewriting, query optimization and transaction management in a front-end process, while query execution, access control and recovery management are handled by the back-end. Front-end and back-end communicate using an intermediary language called Monet Interpreter Language (MIL), and many front-ends can be implemented. Among the currently available front-ends we can list SQL and XML/XPath.

Extensive use of OS services: Monet tries to avoid duplication of code between the OS and the DBMS. Some OS services used by Monet include: multi-threading and scheduling, file I/O, virtual memory buffers and memory mapped I/O.

3 Benchmark

Before anything, we have to determine a meaningful metric for measuring the performance and select a representative workload to comparatively assess Monet, MySQL and PostgreSQL for the construction of dynamic web applications. Read-optimized databases are most commonly compared using the TPC-H benchmark [15] for decision support and business intelligence [3, 7]. However, we want to explore the behavior of these databases for web applications. In this case, the TPC-W benchmark [13, 8] is the best choice. It defines web interactions per second (WIPS) as its main metric and is judiciously precise in the definition of the workload. TPC-W has been put to test by different groups in academia and industry, and its success has granted it the reputation of industry standard [1, 4, 5, 6].

Approximately two years ago, the TPC has created an even more demanding and detailed benchmark for web applications, it is called TPC-App [14]. It allows the assessment of complex web applications and services, based on different communication protocols, including message oriented middleware. The implementation of TPC-App is dearer than the implementation of TPC-W, in terms of human, software and hardware resources. To this date only a major IT company has implemented it, meaning that TPC-W is still the *de facto* standard for benchmarking web applications. The rest of this section describes the features of TPC-W that are relevant to this research, including a detailed explanation about the particular TPC-W implementation used in the experiments.

3.1 TPC-W: Workload and Metrics

The TPC-W benchmark simulates the workload of a web book store. This is recognized as a very typical e-commerce application, with behavior representative of a large class of dynamic content web applications. The TPC-W specification [13] stipulates

the exact functionality of this web store, defining the access pages, their layout and images (thumbnails), the database structure and the client load. However, the TPC does not provide a reference implementation of the web book store, the benchmark definition is careful to the point of guaranteeing that implementation differences do not impact on the benchmark results. The workload load is implemented by remote browser emulators (RBEs), that navigate the web book store pages according to a customer behavior model graph (CBMG) [8]. A CBMG describes how users navigate through the interface of a dynamic web application, which functions they use and how often, and the frequency of transitions from one application function to another. Thus, a CBMG is a useful tool for the definition of workloads for web applications.

A TPC-W run can generate one of three defined workload profiles. Workload profiles are differentiated by varying the ratio of browsing (read access) to ordering (write access) web interactions, and each one has its corresponding metric. An example of a read-only access in the CBMG is the search for books by a specified author, while an example of an update access is the placement of an order. The basic shopping profile, with the general WIPS metric, generates a mixed workload of browsing and ordering web interactions. The browsing profile, with the WIPSb metric, generates a primarily browsing workload. The ordering profile, with the WIPSo metric, generates a primarily ordering workload. More precisely, the shopping profile stipulates that 80% of the accesses are read-only and that 20% generate updates. The browsing profile stipulates that 95% of the accesses are read-only and that only 5% generate updates. Finally, the ordering profile stipulates a distribution where 50% of the accesses are read-only and 50% generate updates. The implementation of these profiles is made in the RBE by specifying the transition probabilities for the CBMG.

TPC-W also has very strict rules for the database schema and for the type and amount of data generated to populate the relations. Specifically, the amount of data stored is used to test the scalability of the system under test. There is a scale factor defined as a function of the number of RBEs and products stored in the system. As the number of RBEs increase, so does the number of user profiles and orders stored. As the number of items increase, so does the number of authors. Table 1 summarizes the scale relationships among the various data relations as defined by TPC-W.

The workload generated by the RBEs emulates the interactions between human clients and the web book store. As the pattern of interactions generated by real users tend to alternate actual requests (interactions) with periods of inactivity, the workload mimics the real pattern by interspersing interactions with delays called think times. On average the think time is defined by TPC-W as 7 seconds, and at any given time, the total number of web requests being generated per second by the set of emulated RBEs can be calculated as the (Number of RBEs) / 7.

Table	Number of Rows
CUSTOMER	2880 * Number of RBEs
COUNTRY	92
ADDRESS	2 * CUSTOMER
ORDERS	0.9 * CUSTOMER
ORDER_LINE	3 * ORDERS
CC_XACTS	1 * ORDERS
ITEM	1k, 10k, 100k, 1M, 10M
AUTHOR	0.25 * ITEM

Table 1: Database size as a function of number of RBEs.

3.2 TPC-W: Implementation

As mentioned, the TPC-W benchmark does not offer a reference implementation, as many components of a complete implementation are indeed part of the system under test. This generality of the benchmark is a positive characteristic, as it allows for the evaluation of systems implementing very diverse software and hardware architectures. However, this puts a considerable implementation burden on the execution of the benchmark as many business logic and integration components must be built and integrated. This is specially true when the performance of just a sub-system of the whole system is the focus, as it is the case of this work where the focus is the database (third tier of the web application).

As we are only interested in the performance of the database deployed with the web application, instead of implementing the book store from scratch, we selected an existing implementation of TPC-W and retrofitted it with different databases. The implementation used was originally developed by a research group of the University of Wisconsin-Madison and is available at the PHARM project web site[4] [4]. This implementation is an accurate and almost complete rendition of the TPC-W benchmark for the JEE platform. The benchmark was written for the assessment of a DB2 database, so we had to make its implementation database transparent through the modification of the JDBC database connection abstraction layer. We strove to make sure changes were kept minimal and restricted to type conversions, and to the adaptation of date and numeric literal formats.

The implementation obtained after we applied our minor changes implements accurately the TPC-W benchmark with respect to the book store application, data layout and workload specification. Despite this, it is important to stress that our implementation and the original are just partial implementations of the benchmark. Thus, the benchmark results obtained in our experiments should not be compared to

[4] http://www.ece.wisc.edu/~pharm/tpcw.shtml

the official TPC-W results. In particular, our implementation has the same limitations of the original implementation; a non-comprehensive list of the simplifications include:

- The remote payment emulator has not been implemented.
- Query caching has not been implemented.
- The image server, used to load and store thumbnails of book covers, is the same server used for the application server.
- There is no secure socket layer support for secure credit card transactions.
- Logging of application messages for audit purposes has not been implemented.

Despite this restrictions, the benchmark implemented is able to emulate with high accuracy the patterns of web interactions specified by TPC-W. As important as the ability to generate the correct workload, is the fact that all three databases are indeed subject to the same workloads, guaranteeing the fairness of the measurements.

4 Experimental Setup

The experimental setup included the following versions of the database managers:

- Monet 4.10.2
- MySQL 5.0.15-max
- PostgreSQL 8.1.3

All databases were downloaded from their official distribution, configured and compiled using only their default settings. The application server used was Apache Tomcat 5.5.15 running on Sun Java 1.5. The application server and the databases were setup and run in separate machines, with communication achieved via JDBC, employing TCP/IP connections. Due to limitations of the JDBC/SQL front end of Monet, all databases were configured to use fully serialized transactions and the JDBC connection polling function was deactivated. The use of fully serialized transactions affected uniformly the performance of the databases compared, without any negative effect on relative performance figures. The fact that only this mode of transaction isolation was available for Monet was an early indication of the problems we faced to interface it to Java applications.

The hardware setup comprised two machines: Host A runs the RBEs and the application server and Host B runs exclusively the database under test. The setup provides isolation between the database server and the application server, guaranteeing sufficient hardware resources were always available for the DBMSs. Table 2 lists

Host	Configuration
A	Pentium 4, 2.8 GHz, 1GB RAM, Fedora Core 5
B	Dual Pentium 4, 3 GHz, 2GB RAM, Fedora Core 5

Table 2: Hardware configuration.

the hardware configuration of the two machines used. Both hosts were interconnected by a 100Mbps switched Ethernet link.

The databases were populated according to the rules specified by TPC-W (Table 1), using a 10000 items scale factor and 30, 150 or 300 RBEs. We used these three numbers of RBEs to finely adjust the size of the database, instead of increasing the items scale factor. In some of the tests performed, a smaller or larger number of RBEs was effectively used compared to the number used to populate the database. The several load setups used in the tests are listed in Table 3.

	Database Populated With	Tested With
Small	10000 items, for 30 RBEs	30, 150, 300 and 600 RBEs
Medium	10000 items, for 150 RBEs	150 RBEs
Large	10000 items, for 300 RBEs	300 and 600 RBEs

Table 3: Load setups tested.

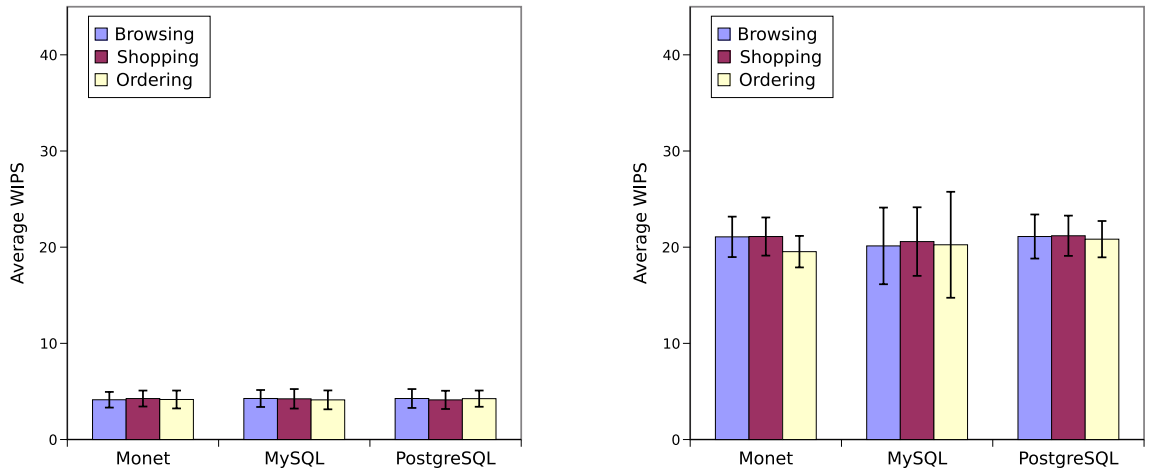
The small setup has few clients loaded into the database and was designed to assess the performance when all data is resident in the main-memory of the DBMSs. The other two setups stress the system by scaling up the number of web interactions generated by RBEs and database sizes. The largest database was limited to data for 300 RBEs because Monet was not able to populate a larger database in a reasonable time. Also, due to the time required to populate all databases, population was made only once and backups were made using the tools available for each DBMS. Before each test run, the backups were restored ensuring that all tests were performed with a newly populated database. Each test run extended for 10 minutes, with a 1 minute ramp-up time. We collected data for all the three metrics defined by TPC-W, with results presented as the average WIPS, WIPSb and WIPSo measured during the 9 minute stable execution interval. An important aspect of the tests is that the recomendations of the TPC-W for the reproducibility of tests have been followed [13, pp. 84-108]. These recommendations forbid, among other restrictions, the redefinition of the database schema between tests, changes to the physical placement and/or distribution of the data, changes or reboot to any version or different type of any of the software modules installed originally, etc.

5 Results and Analysis

This section contains de results of the comparison, grouped by database size (Table 3). For each group of results data is summarized by displaying the average number of web interactions per second (WIPS) and standard deviation from the average for the stable sampling interval.

5.1 Small Database

The small setup ensures that the entire database fits in main-memory. This setup should show us how well the databases manage their use of RAM and how well the main-memory optimizations of Monet perform. Figure 2 shows the data for a small number of RBEs, 30 and 150. In both cases all databases were capable of handling all web interactions that were generated by the RBEs, performing at approximately 4 and 21 WIPS. There is a very small difference among the ordering, shopping and browsing profiles and the small deviation indicates a sustained throughput. The results obtained ensure us that all three DBMSs are working correctly because the average WIPS matches the load generated by the RBEs (see section 3).



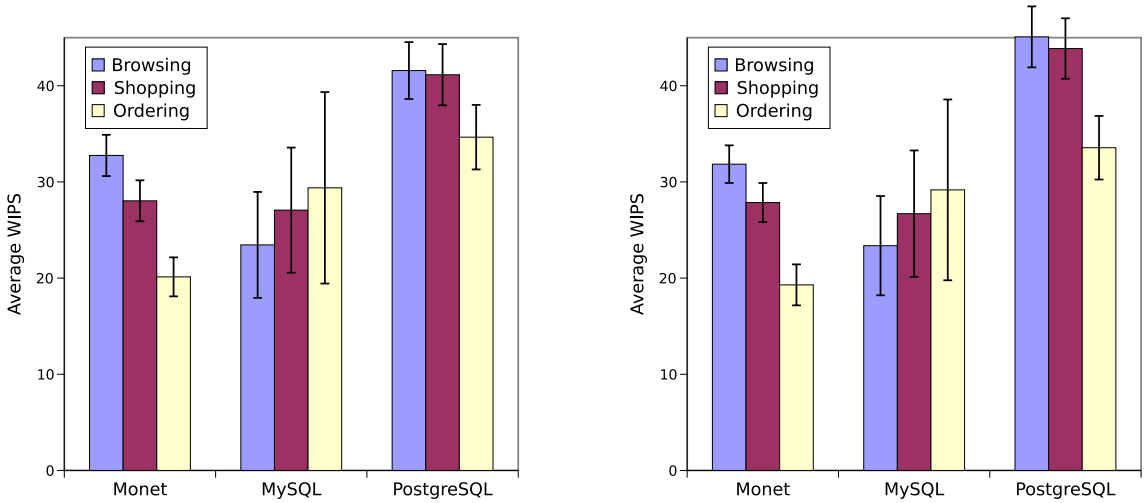
(a) 30 RBEs

(b) 150 RBEs

Figure 2: Average WIPS with data for 30 RBEs.

Increasing the number of RBE clients stresses the transaction processing engines of the databases and noticeable differences emerge as shown in Figure 3. The only database capable of handling the workload generated for at least one of the profiles (Figure 3a, WIPSb 300 RBEs) is PostgreSQL because the measured WIPS fall within

the expected values of WIPS (42.85 WIPS). Monet and MySQL fall behind PostgreSQL in requests served, but Monet is better than MySQL by a narrow margin in the browsing and ordering profiles. However, as expected Monet shows a significative performance drop for the ordering profile; this is predictable as it is read-optimized and this profile has a read/update ratio of 1 (50% read/50% update). When attention is focused on the deviations, results show very small standard deviations for Monet and PostGresSQL indicating their capacity to execute the workload as prescribed by TPC-W. Differently, MySQL has started to show greater variability of WIPS. A feature worth of note is the performance of MySQL when compared to itself for the three different profiles: WIPSb is smaller than WIPS that is smaller than WIPSo. This implies that MySQL increases its performance as the ratio of reads to updates increases, a surpring and unexpected behaviour because updates imply disk writes and disk writes imply higher latencies and higher latencies should imply smaller WIPS values. Monet and PostgresSQL behave as expected, that is, show smaller values of WIPS as their results progress from the browsing profile the ordering profile (WIPSo).



(a) 300 RBEs

(b) 600 RBEs

Figure 3: Average WIPS with data for 30 RBEs.

5.2 Medium Database

The medium setup is expected to stress the I/O capacity of the DBMSs, as the data does not fit in main-memory; it should be critical to Monet as it is not only read-optimized but also a in-memory database. Figure 4 shows that in this setup only PostgreSQL and Monet were able to handle all requests generated by the RBEs.

PostgreSQL once again was the best, displaying a performance very close to the expected 21 WIPS in all profiles. This implies that it was still able to handle the larger database using main-memory. The same can be observed for Monet, it is able to handle the expected 21 WIPS for the shopping and browsing profiles. However, in comparison to its performance for smaller database setups, the performance drop was much larger for the ordering profile; it was able to deliver only 9 WIPS. MySQL could not sustain the expected 21 WIPS for the read dominated profiles, but was able to display the same unexpected good performance in the update dominated ordering profile. The WIPS standard deviations have increased significantly for MySQL, indicating that it may reach a limit after which it will not be able to respond to the workload.

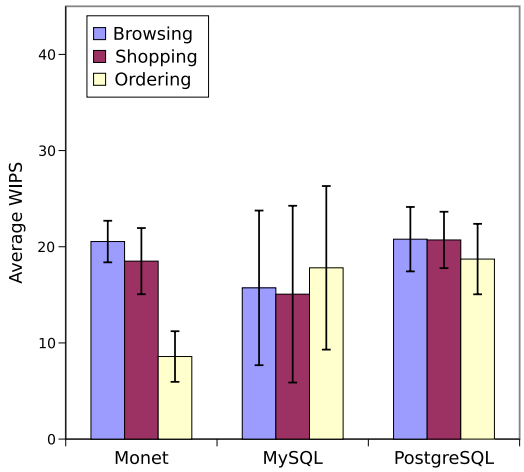
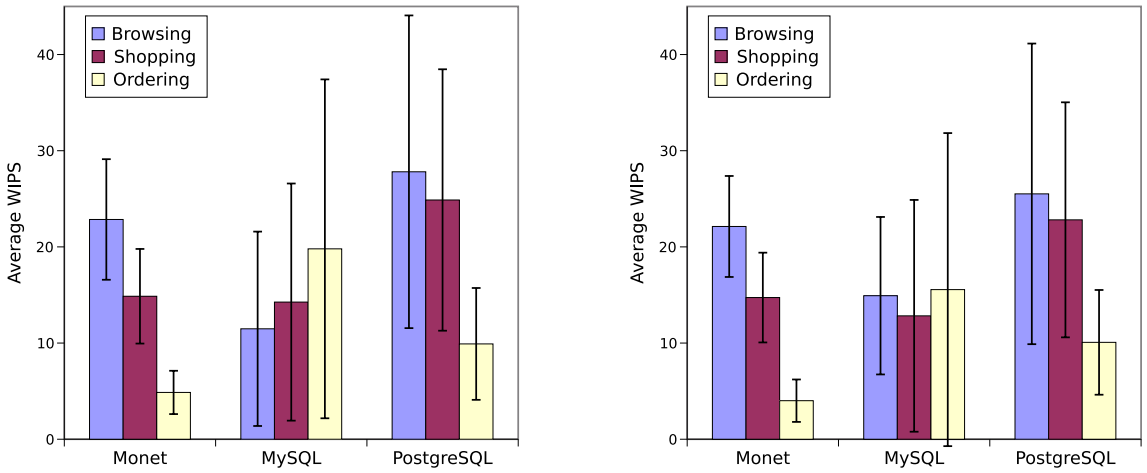


Figure 4: Average WIPS with data for 150 RBEs.

5.3 Large Database

The large database can't fit in the main-memory of the analyzed DBMSs, thus this setup puts the most strain in the I/O subsystem of the databases and a very noticeable drop in performance occurs for all three databases shown in Figure 5. PostgreSQL is still the best performer but it can't keep up with all the requests generated by the RBEs. Also, it is possible to notice a very sharp drop in performance for the ordering profile. Monet comes in second in the browsing and shopping profiles. An interesting situation is observed once again, MySQL in update-intensive ordering profile is almost as good as Monet in the read-intensive browsing profile. The Monet behavior is expected, and is a direct consequence of the read-optimization of this DBMS. MySQL confirms its surprising feature of displaying better performance as the proportion of

updates increases. At this stage in of the experiments, our explanation for this fact is that MySQL isn't making the updates durable when the transaction commits, it probably just caches updates and committs them to disk opportunistically when it detects a load decrease. In this scenario, the variability of the performance shows an interesting feature of the databases. Monet and PostgresSQL are not able to sustain the workload, but show a smaller variability, specially in the ordering profile. The results show that MySQL is not only unable to sustain the workload but also that it is subject to much larger deviations, meaning that it is observed by its clients as an erratic DBMS that stalls transactions when heavily loaded.



(a) 300 RBEs

(b) 600 RBEs

Figure 5: Average WIPS with data for 300 RBEs.

5.4 Discussion

For the small database, all DBMSs behaved in a very similar way. Only when the load or the database size are increased is that their relative performance started to diverge, with PostgreSQL being the clear overall best performer. MySQL displayed a very solid performance in the ordering profile, but with an erratic behavior, assessable by the magnitude of the WIPS standard deviations. Monet displayed a performance as good as the other DBMSs, but with an expected performance drop in the update-intensive ordering profile. Considering that MySQL and PostgreSQL are very popular as back-ends for web applications, the results show Monet as a good platform for the application domain considered. However, it wasn't 10 times fast as expected, not even in the browsing profile, and exhibited several implementation problems. The SQL front-end was prone to crashes and the JDBC interface was incomplete and had

some minor bugs. Even considering the performance penalty incurred by frequent updates, the process of populating the database prior to running the tests was very time consuming. It took about 24 hours to load the data for 300 RBEs, and we gave up after 48 hours of wait for the creation of the database for 600 RBEs; for this reason we were forced to limit the largest database used in the experiments to a database populated for 300 RBEs.

Despite these problems, the performance of Monet was acceptable but not exceptional. It performed in the same order of magnitude of the other, more mature, DBMSs. If we assume dynamic web applications are a domain that has a great similarity with OLTP applications, and considering both MySQL and PostgreSQL are heavily optimized for this class of applications, then one would expect the performance of Monet to have been comparatively worse. If, alternatively, dynamic web applications share the transaction profile of OLAP applications, Monet should have displayed an even better performance than the observed, at least in the read-intensive browsing profile. If order of magnitude performance improvements are to be obtained, then new database management designs have to be pursued.

6 Conclusion

In this paper, we investigated the use of a specialized DBMS for data warehousing as data back-end for web applications and assessed the performance of this combination of database and application. We performed benchmarks to compare the performance of Monet with two of the most commonly used databases for web applications: MySQL and PostgreSQL. Our results show that, at least for now, it is not possible to obtain 10 fold gain in performance using Monet as a simple drop-in replacement for a standard DBMS in a typical web application configuration using Java and JDBC.

Monet was as fast as the general purpose databases, but not faster. However, it still has room for improvement as it is a relatively young project compared to MySQL and PostgreSQL. After the time we did our tests, the Monet team has released new versions of their database engine, with improved performance and better stability. Moreover, the German IT magazine c't invited database companies for a comparative performance competition where a solution created and optimized by the Monet creators showed the best performance for Java based solutions [9]. Nevertheless, the fact that both OLTP and OLAP optimized databases displayed comparable performance for the domain of dynamic web applications, allow us to question a widely accepted notion that RDBMSs are the best solution for the data back-end of web applications. Our experimental results provide for the first time a quantitative indication that web applications may have their own specific requirements for data management and may constitute a rich niche for the development of specialized databases.

References

[1] Cristiana Amza, Anupam Chanda, Alan L. Cox, Sameh Elnikety, Romer Gil, Karthick Rajamani, Willy Zwaenepoel, Emmanuel Cecchet, and Julie Marguerite. Specification and implementation of dynamic web site benchmarks. In *WWC-5: Proceedings of the IEEE International Workshop on Workload Characterization*, pages 3–13, November 2002.

[2] Peter A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[3] Peter A. Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-pipelining query execution. In *CIDR 2005: Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, pages 225–237, January 2005.

[4] Harold W. Cain, Ravi Rajwar, Morris Marden, and Mikko H. Lipasti. An architectural evaluation of Java TPC-W. In *HPCA '01: Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 229–240, Monterrey, Mexico, 2001.

[5] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. C-JDBC: Flexible database clustering middleware. In *Proceedings of USENIX Annual Technical Conference, Freenix track*, 2004.

[6] Daniel F. García and Javier García. TPC-W e-commerce benchmark evaluation. *Computer*, 36(2):42–48, 2003.

[7] Stavros Harizopoulos, Velen Liang, Daniel J. Abadi, and Samuel Madden. Performance tradeoffs in read-optimized databases. In *VLDB 2006: Proceedings of the 32nd international conference on Very large data bases*, pages 487–498. VLDB Endowment, 2006.

[8] Daniel A. Menascé. TPC-W: A benchmark for e-commerce. *IEEE Internet Computing*, 6(3):83–87, 2002.

[9] Hajo Schulz. Entscheidende maßnahme – c't-datenbank-contest: Die auflösung. *c't Magazine*, 13:190–193, 2006.

[10] Michael Stonebraker, Chuck Bear, Uğur Çetintemel, Mitch Cherniack, Tingjian Ge, Nabil Hachem, Stavros Harizopoulos, John Lifter, Jennie Rogers, and Stan Zdonik. One size fits all? Part 2: Benchmarking studies. In *CIDR 2007: Proceedings of the Third Biennial Conference on Innovative Data Systems Research*, January 2007.

[11] Michael Stonebraker and Uğur Çetintemel. "One size fits all": An idea whose time has come and gone. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 2–11, Washington, DC, USA, 2005. IEEE Computer Society.

[12] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. C-Store: a column-oriented dbms. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.

[13] TPC. *TPC Benchmark W Specification*, February 2002.

[14] TPC. *TPC Benchmark App Specification*, February 2005.

[15] TPC. *TPC Benchmark H Specification*, October 2006.