A comparative analysis of the LAMP (Linux, Apache, MySQL and PHP) and Microsoft .NET (Windows XP, IIS, Microsoft SQL Server and ASP.NET) Frameworks

Abstract

This paper examines the Microsoft ASP.NET framework version 2 and the LAMP (Linux Apache MySQL and PHP 5) web development platforms.

Findings are drawn from practical experience in developing a large-scale web-based application using both platforms. In addition, preliminary tests have been run across both frameworks in order to establish tentative performance benchmarks.

This paper suggests that although LAMP is a good production environment, .NET 2 is better suited to large-scale, critical enterprise development, while LAMP is better suited to smaller or medium-sized applications or applications which are potentially cross-platform.

1. Introduction

This paper broadly outlines a comparative analysis of the two most popular web-development frameworks; ASP.NET and LAMP [Netcraft]. Section 2 discusses the methodology used to compare the platforms. Section 3 provides an overview of the application, while section 4 details the implementation of the project, outlining experiences from development in both applications, and some initial benchmarking data. Finally section 5 draws conclusions from results to date.

2. Methodology

In order to develop a deep understanding of the strengths and weaknesses of these development environments, large-scale applications were developed using both the LAMP and ASP.NET frameworks. Development experiences as well as tests and analysis of the final products formed the basis for comparative analysis of the two frameworks.

3. Overview

www.38.co.za is an existing, live photo gallery website which was developed by the author The site generates over a million hits a month and to date has over 17 000 photographs, over 200 galleries and a subscribed user-base of almost 6000 users. www.38.co.za serves as the perfect test-bed for testing the strengths and weaknesses of the frameworks because of both the performance and feature requirements of the site.

The application requires efficient and complex data access to feed information from the extensive database and to allow statistical analysis of this data – this tests both large and complex database queries. Security needs to be leveraged to implement authentication and personalization as well as to prevent malicious activity such as SQL injection or XXS (Cross Site Scripting) [Howard, H. *et al*]. RSS, XML and Web Services are used for client side-interaction between the website and various client-side GUI components created to ease administration and add value to user interaction. Finally, because the site is image-centric, code-optimization is required to deliver a high-performance website able to deliver bandwidth-intensive content to the browser quickly and efficiently.

Thus the application presents a complicated specification which provides a good basis for testing the performance and quality of any web-development platform across a variety of important facets.

4. Implementation

The pre-existing application, created by the author, was completely rewritten and refactored.

From the existing, largely procedural 2-tier PHP4/MySQL code: two new versions were created; one using PHP5/MySQL 5 running on Apache 2.0 and the other using ASP.NET 2/SQL Server 2005 running on IIS.

4.1. Architecture of the new system

The architecture of the new, refactored application embraces an Object-Oriented (OO) approach and provides much neater, more elegant, maintainable and extensible code.

Table 1 - Performance comparison

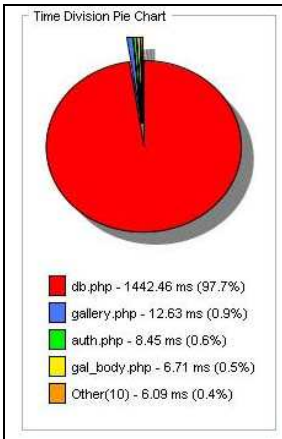

Figure 1 - Performance of the old galleries page

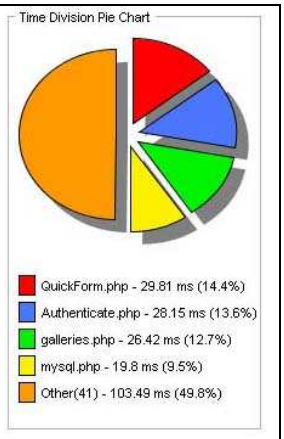

Figure 2 - Performance of the new galleries page

Although a more complex architecture, the new 3-tiered application improved raw performance of the PHP application by as much as 300% on some pages. For example, the galleries page, which handles browsing, searching and listing of available galleries, improved from 1527.07 ms loading time to only 217.59 ms. The diagrams in Table 1, show how the majority of processing time (1442.46 ms) in the old page (Figure 1) was occupied by database calls.

Much of the bloat in the old system was the result of ad hoc additions or changes to code. As a result code was inefficient and in many cases there was unnecessary repetition of SQL queries or logic layer code.

Refactoring the old system not only cut out much of this spurious code - hence the much improved performance - but the new 3-tier architecture features better code separation, object re-use and abstraction to ensure that future ad-hoc maintenance will not result in the inefficiency problems explained above.

By abstracting logic and database calls to the BLL (Business Logic Layer) and DAL (Data Access Layer) respectively, the developer was able to simplify code and logic in the UI (user interface), hence easing maintenance at the UI level.

By abstracting internal logic (such as searching logic) to the BLL, and data queries to the DAL, the galleries UI page was reduced from 579 LOC (Lines Of Code) to only 73 LOC containing the core logic required for that page. This allows for easier-to-read code as well as centralized design and object re-use.

4.2. Developing with LAMP

4.2.1. PHP 5

Unlike ASP.NET, the PHP 5 scripting environment is a lightweight environment encompassing a bottom-up approach to development, providing a minimum of tools within the default installation.

The advantages of this, over complicated frameworks such as .NET are that it provides a simple development platform. With PHP 5 there is no need to learn a large library of objects but it still provides the flexibility to add any necessary objects to the project as and when required [Fuerks]. This enables the developer to get started in this environment quickly, using whatever coding technique they are most comfortable with, while enabling more advanced programmers to use the advanced OO features available as they require them.

PHP is an open-source project, so extensions to the basic language typically come in the form of third-party tools created by community members or groups. A number of ‘frameworks’ have been built on top of the basic PHP engine. The most commonly used one is the PEAR (PHP Extension and Applications Repository)

library – which is closely linked with core PHP development. This is the framework which was used for development of the new system.

An advantage of such a minimalist approach is that it is possible to pool all the class libraries to find exactly what is needed. PHP by default provides a simple framework for development, suited to ad hoc small scale development, however, the language itself supports the OO needs for large-scale development and, when these are needed by the developer, the required tools are available and easy to use and incorporate. This is illustrated by the needs of the target application, www.38.co.za. The initial code was mostly procedural, with a few general-purpose custom classes. As the application has matured, a need for more mature code has emerged and the new implementation takes advantage of the PEAR libraries and advanced Object-Oriented coding practices to produce this.

While the community–based, bottom-up development process is flexible in that it affords the user the opportunity to pick and choose, it has a number of detrimental consequences, especially when large-scale applications are being developed. These include:

4.2.1.1. Lack of uniformity across applications.

The bottom-up approach means that across both applications, and development projects there is a lack of continuity and solidarity. For example: during implementation, design time decisions had to be made as to which framework to use (PEAR, Eclipse, Prado, Smarty, or a compilation) for development. These decisions would greatly effect the future development of the application.

While this afforded flexibility to the project in terms of the ability to use frameworks with which the developer was comfortable, it also led to confusion as to which framework *should* be used, and to potential future time wastage as developers new to the project may need to learn a new framework.

Every PHP project is unique. This makes it more difficult for newcomers to a project to understand its intricacies. The greater choice of core components (be it class libraries, templating engines or even patterns subscribed to), and a lack of a standard core set of components, means that there is a lack of uniformity in applications developed using PHP.

Flexibility has been a great aid in promoting PHP as the best platform for developing smaller-scale applications [Berkes, D.], but unfortunately the features responsible for the flexibility also limit its entrance into the domain of large-scale enterprise development.

4.2.2. Questionable quality of third-party tools

Although certain tools and libraries are generally accepted by the community, there is still no guarantee of quality from third-party classes.

Even using PEAR, the generally accepted standard library, the developer had problems with poor documentation and classes which hadn't been updated in a long time; or for which production had ceased. This complicated the process of development with this library, and also raised questions as to the future stability of packages used.

Although the core packages in the library (such as DB or HTML::Quickform) were found useful and well documented, the non-core packages (such as DB_SLIDER) were undesirable due to poor documentation and questionable quality.

Furthermore, the unintuitive structure of the library also led to uncertainty. For example: the DB_SLIDER class (which provides paging functionality to query results) is packaged in the Database package, instead of the HTML package where the other PAGER objects are packaged.

In addition, many classes in the PEAR repository are very specific, and are carelessly

implemented as stand-alone classes rather than classes which inherit from a more generic base class. This limits the flexibility of these classes. A class such as PEAR_AUTH (which handles user authentication) is an example of this. It was deemed simpler to write a custom class to handle authentication, rather than try to plumb the PEAR_AUTH mechanisms into the existing project.

Simple issues like this, when compared to the careful structures of interface-driven frameworks like .NET and Java, cause even the best third-party PHP libraries to pale in terms of functionality and usability. While a successful framework (such as .NET) needs careful planning and design, the PEAR 'framework' has been developed in a largely ad hoc manner and this is noticeable from its lack of structure, documentation and its somewhat erratic nature.

As has been stated, the major barrier to PHP's entrance into the enterprise is its bottom-up nature. Whereas monolithic frameworks such as .NET or Java provide an entire framework for core development, PHP has not yet established a strong framework for development, possibly due to the fact that it has only fairly recently embraced Object Oriented (OO) coding practises, and PHP 5 is arguably the first properly OO implementation of PHP. This lack of adequate and uniform design tools hampers PHP's entrance into the enterprise.
At the enterprise level of development, issues such as team development, uniform design and quality of components become of critical importance – these are issues which PHP still needs to improve in order to compete with the major frameworks such as .NET and Java, which have extensive native libraries as well as strong documentation and established design patterns of best use [Thilmany, C. pp: 35-39].

4.2.2. MySQL

Similar to PHP, MySQL is an open-source, lightweight database engine, renowned for its high speed [Troels].

Its high speed makes MySQL ideal for web development where response-time is often of critical importance. Another nice feature of MySQL is its support of multiple table types. This feature was useful for optimizing the application. While the majority of tables in the database were of type InnoDB (which enables relationships and integrity checking of data), there were performance advantages in changing the table type to the faster MyISAM for stand-alone or large tables such as the countries or statistics tables. This trick however should be used carefully as MyISAM tables are fairly 'dumb', and offer no integrity checks. As such MyISAM tables were used sparingly in the application.

The major short-coming of the MySQL database server was its limited feature-set. In particular the lack of support for stored procedures (only available in the MySQL 5 BETA version), meant that more PHP code needed to be written at the DAL. For the .NET application, however, most if not all SQL could be pushed into the SQL Server DBMS (Database Management System) in the form of stored procedures (which are faster and more secure [Howard, H *et al*]), this could not be deemed a best practise in the LAMP implementation due to the new and unstable nature of this feature in MySQL.

4.2.3. Apache

Apache is in many ways the strongest tool in the LAMP toolkit. The major strength of Apache, apart from its majority market share and strong security record (which is discussed later) is that it is a solid cross platform web server. This means that LAMP applications can run along-side Java applications on a UNIX server, as well a alongside .NET applications on a Windows machine.

Apache's modular design makes it extensible and also supports a rich feature-set. A major advantage leveraged in the application was the use of the MOD_REWRITE module to enable search engine friendly urls using regular expressions. Using MOD_REWRITE; urls

could be changed from the complex http://www.38.co.za/gallery.php?id=6&pic=456 to the more user-friendly http://www.38.co.za/gallery/123/456. This ad introduces a certain level of security through obscurity to the application.

4.2.3. Conclusion

In conclusion the lightweight LAMP environment seems typically better suited to smaller problems. The lightweight package affords simplicity to simple problems, but ironically the inherent simplicity of the framework adds complexity to solving larger more complex problems.

In the hands of an experienced developer the LAMP environment is a powerful and cheap tool for development, and is capable of elegant solutions to complex problems (such as the new implementation of www.38.co.za). However, the tool is not ideally suited to this type of development, and works better at easing the development of poor and hard-to-maintain applications (such as the old implementation) in the hands of a novice developer.

4.3. Developing with the .NET framework

The .NET framework, now in version 2, represents a mature development framework which attempts, with high levels of success, to provide the user with all the necessary tools for development.

.NET provides an extensive component-based system which encapsulates a great deal of the standard requirements for a web–based application, such as: data access and display; user authentication and personalization; session maintenance; and input validation. The built in management of these fundamental requirements means that the developer is able quickly and easily to develop a skeleton application and therefore is able to concentrate on application-specific value-added tasks and logic. .NET speeds the development process by reducing the need to re-invent the wheel every time a new application is developed.

For example: during development of the LAMP application much time was dedicated towards mundane tasks such as creating data objects [Patterson, D.] for the DAL to represent an in-memory representation of the database. Further time was spent writing classes to perform common BLL and presentation layer functionality (such as authentication and database paging), because such classes did not exist or were of poor quality.

By contrast, the tools provided by ASP.NET and the .NET framework made these tasks simple and quick to perform: Datasets [Thilmany, C. pp: 204-217] were used to provide an efficient and flexible in-memory persistent storage map; controls such as the login and personalization controls, as well as databound display controls, such as the datalist and datagrid, enabled quick and easy development of the skeleton of the application, enabling the developer to concentrate on the specific unique business logic required, such as dealing with adding galleries, comments and franchises.

A common difficulty with component-driven architecture is that often by providing specific functionality to components, it compromises their flexibility. .NET however, using the provider pattern, as described by [Howard, R.], and an interface-driven framework has successfully created a very extensive, powerful yet very flexible development environment which allows users to use controls either as they come; slightly altered; or completely rewritten as custom providers.

During implementation this flexibility was leveraged to provide a custom membershipProvider class for login controls on the application [Ghosh, J.]. This enabled the use of the .NET login controls using the existing table structure containing the user information from the initial application.

By comparison, the authentication features provided by the PEAR library are limited in both functionality and flexibility. Ultimately for the functionality offered by the PEAR_AUTH package, it was more time-efficient to create custom classes then to use those provided.

The ASP.NET environment is a complete and polished development environment, providing not only a solid framework with which to develop, but also numerous useful tools, such as the new test suite in Visual Studio 2005, which ensure the development of quality software.

The major short-coming of ASP.NET is its reliance on the Windows/IIS combination. In particular dependence on IIS is very undesirable. Not only is Apache deemed a more secure server [Varner, P], but it also dominates market share at 69.46 % compared to IIS's 20.43% [Netcraft]. This makes finding a good host for .NET applications slightly more difficult and typically slightly more expensive (this is also obviously due to the fact that the LAMP stack is all open source and free software).

Platform reliance also effects interoperability of the application within an organization. While LAMP can run on Apache opposite Java or Perl applications as well as on IIS opposite .NET applications, .NET is limited to the Windows/IIS environment. This is a limiting factor for Linux-house companies.

4.2.2.1. Conclusion

The .NET environment is a complete and polished development environment. Providing not only the framework, but also the tools (Visual Studio, Visual Web Developer or Web Matrix and Quality testing tools provided by the test suite) to ease the production of high-quality large-scale web applications. An extensive framework of patterns and a carefully implemented framework make the development of high-performance elegant solutions with .NET easier than ever.

Although the .NET environment is one of the friendliest development environments existing, its major short-coming is platform dependency. Although through Web Services and the .NET framework interoperability has been greatly improved, the framework still ties the user to a Window's platform.

4.3. Performance

To date no formal testing has been performed as implementations have not yet been deployed to valid testing servers. Results discussed below serve only to give a general impression of performance.

Tests were conducted on the development computer – a Pentium 4, with 1 gigabyte of RAM. Tests were conducted in a Windows XP environment, and this may have affected the performance of the [L]AMP installations.

The WAMP (*Windows*, Apache, MySQL and PHP) environment consisted of Apache 2.0 server (with Zend optimizer), MySQL 5 BETA and PHP 5. The .NET environment entailed IIS, SQL Server 2005 BETA April CTP, and .NET framework 2.0 BETA.

Response time tests were collected using in-code timing, the Zend Profiler tool, and Web Tests writing in Visual Studio 2005. Load tests where also performed using Visual Studio 2005.

Table 2 - Response times for ASP.NET and LAMP

Timing tool	ASP.NET	LAMP
In-code timing	N/A	1.187
Zend Profiler	N/A	2.458
Visual Studio web tests	0.45	0.95

- Times are an average of 10 timings taken
- All times are in seconds

As can be seen, the results in Table 2 suggest faster response times from the ASP.NET code.

This can very likely be attributed to the improved efficiency brought about by the use of compiled code.

5. Conclusion

In conclusion, in all sections regarding pure development ASP.NET proved to be a favourable environment to the LAMP. ASP.NET provides better, more elegant and easier to implement solutions for handling data access (datasets versus hand-coded php data objects), XML and Web Services (Visual Studio provides excellent tools to facilitate the entire process from discovery to deployment simply and elegantly), Security and Optimization. Although the LAMP is capable of achieving all that can be done with .NET, solutions of equal or even inferior quality typically required more effort, thought and time.

Where LAMP shines is in terms of interoperability, and cross-platform solutions. Whereas .NET can run only on Windows machines with IIS, each of the LAMP components (barring of course Linux) is able to run on almost any platform: Linux can be swapped for Windows; Apache can be swapped for IIS; PHP can be swapped for Perl, Ruby or Java; and MySQL can be swapped for PostgreSQL, Oracle or SQL server. The strength of LAMP lies in its flexibility – an application produced in LAMP affords the enterprise flexibility within their development environment and does not tie the enterprise into a Windows-centric development environment.

In the opinion of the author, .NET is the overall better development environment. However, the LAMP environment, in situations where the enterprise adopts or is likely to adopt cross-platform solutions, is the more practical and flexible of the two.

As a result, .NET solutions should be adopted where complicated large-scale applications are required, *and* the target platform, and the platform for future development within the enterprise is known to be Microsoft. In situations where there is doubt, however, LAMP provides a capable, feasible and desirable option.

6. References

1. Patterson, D. 2004. **Simplify Business Logic with PHP DataObjects**. Onlamp.com. Available at: http://www.onlamp.com/pub/a/php/2004/08/05/dataobjects.html Accessed: 9 Aug 2005
2. Thilmany, C. 2004. **.NET Patterns.** Addison-Wesley. Boston USA.
3. Fuecks, H. 2004. **The PHP Anthology Volumes I & II.** Sitepoint Press. Australia.
4. Howard, M., LeBlanc, D. 2002 **Writing Secure Code**. Microsoft Corporation Press, USA.
5. Ghosh, J. 2004. **Building Custom Providers for ASP.NET 2.0 Membership.** Online. Available at: http://msdn.microsoft.com/library/en-us/dnaspp/html/bucupro.asp Accessed: 04/09/05
6. Howard, R. 2004. **Provider Design Pattern.** Online Available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp04212004.asp Accessed: 02/09/05
7. Netcraft. 2005. **Netcraft Web Server Survey.** Available at: http://news.netcraft.com/archives/web_server_survey.html. Accessed: 24/09/05.
8. Troels, A. **Comparison of different SQL implementation.** Available at: http://troels.arvin.dk/db/rdbms/ Accessed: 25/05/05.
9. Berkes, D. 2005, **Survey says: PHP passes Microsoft Active Server Pages**. Available at: Accessed: 26/05/05