Performance Analysis in Loosely-Coupled Distributed Systems

Rebecca Isaacs, Paul Barham
risaacs@microsoft.com, pbar@microsoft.com

1 Introduction

A great deal of work is currently underway to develop infrastructure for supporting a loosely-coupled computational model over the Internet. This is based on the standardization of *Web Services* interfaces and protocols to allow application-to-application communication and interoperability using XML messaging[1]. An example of an existing Web Service is the Google search API for programmatic access to Google's search service[2]. Another is the Microsoft Passport which maintains an online identity for users that can be accessed by other applications for purposes of authentication[3]. With the introduction of comprehensive interoperability standards, it is expected that Web Services will become more and more common. Consequently deployment aspects which remain significant challenges for the service provider need to be addressed, including, among others, performance, configuration, management, security and dependability. This paper considers some of the performance-related issues which must be tackled in order that the potential for Web Services to provide a truly global distributed system be achieved.

The effective operation of a Web Service requires more than just interoperability between a user (the party invoking the service) and a service. The service provider needs some means of determining that performance is reaching desired levels in order to provision the right amount of computation or network resource or to make Quality of Service (QoS) guarantees. The service user may want to determine whether their request received adequate performance, or, if something has gone wrong, determine at what stage this occurred. Web Services are made up of federated trust domains which involve an arbitrary number of mutually untrusting participants. The provision of debugging and diagnosis tools is particularly challenging in this environment, and there is a compelling case for standardized debugging and performance profiling APIs.

There are two problems in defining a standardized performance API for Web Services. Firstly, system performance is typically regarded as a characteristic of the individual items of equipment, with various performance counters used to indicate the current state and hence the service received by the service invocations currently in progress. This top-down approach to performance monitoring is ideal for the manager of those resources. However it is not necessarily desirable for the user, for whom the 'unit of interest' is the end-to-end performance of an individual *transaction* across possibly many systems and trust domains.

Secondly, observations which are averaged over many requests may not give sufficient information to determine the behaviour of a service invocation in the required detail, for example,

to get a breakdown of time spent in each constituent action. For these reasons, a standardized performance query API for Web Services should support the gathering of per-request performance details.

2 The Magpie Approach

A key requirement for a Web Services profiling infrastructure is the ability to account the resources consumed by a particular transaction throughout its lifetime. This approach is analogous to that of ProfileMe, which profiles processor hardware performance by attributing events to specific instructions[4]. We have developed a performance monitoring tool for Microsoft Windows called Magpie which applies this approach to a typical e-commerce system, and then automatically characterises the workload of the whole system. Magpie works by tagging incoming requests with a unique identifier and associating resource usage throughout the system with that identifier until the request leaves the system. It subsequently applies machine learning techniques to the information gathered in order to construct a probabilistic model which captures the typical behaviour of each type of transaction. In this way, Magpie ties high-level events, such as the arrival of an HTTP request in the web server, or the processing of an SQL request at the database, with very low-level resource usage information, such as context switches, disk IO and network packets sent and received.

The Magpie system is entirely 'black box', requiring no application-specific instrumentation at all. This is for two main reasons. First, since Windows is a proprietary system we could not assume source code access. Second, by adopting a black box approach we have a set of re-usable components which can be used to instrument other system infrastructures. A useful side-effect of this externalisation of the profiling infrastructure is that it gives us the hooks to provide an API supporting transaction-oriented performance monitoring.

The implementation of Magpie involves a number of standard techniques which are stitched together to propagate the unique transaction identifiers to all components of the distributed system, in particular, some library calls are instrumented using the Detours system for binary interception[5], and synchronisation between threads servicing a transaction is captured. Figure 1 shows part of a visualisation of the resource consumption recorded by Magpie for a single web request. Although to date we have only applied the technique to an isolated 3-tier client-server system, in the near future we are planning to profile a peer-to-peer messaging framework, and, as discussed in this paper, we believe that the approach is a good fit for wide-area distributed systems such as Web Services.

3 Predicting Performance

The original motivation for developing Magpie was the needs of a performance modelling toolkit called Indy[6]. A performance modeller using Indy must know the 'cost' of each type of operation in order to generate realistic workload models for input into the performance simulator. Such data is typically obtained using techniques similar to benchmarking, where a metric is specified by which the performance of one system can be compared with that of another. For example, the TPC Web Commerce Benchmark (TPC-W) records the number

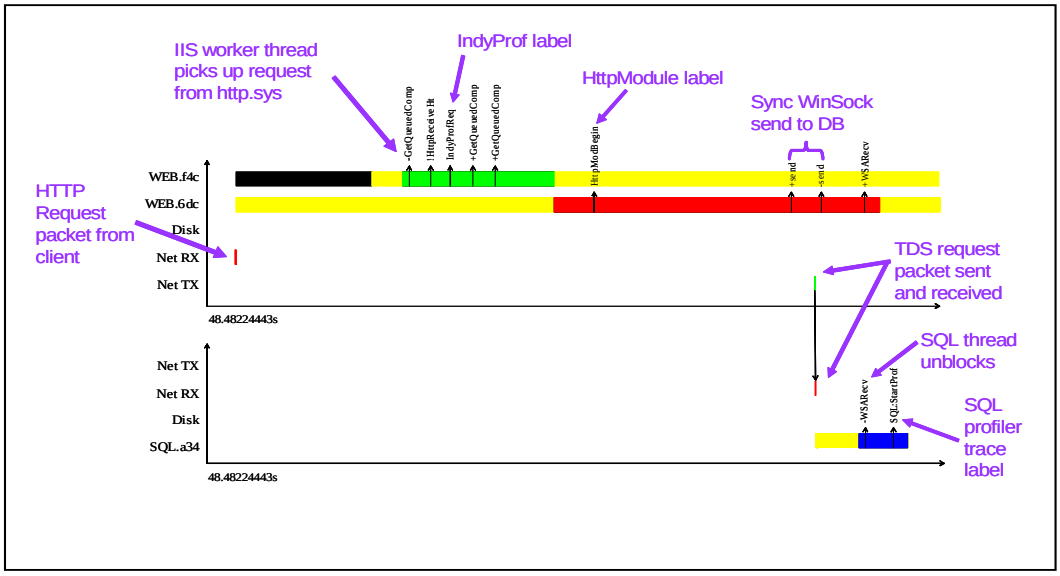

Figure 1: Part of the annotated visualisation of the resource consumed by a single HTTP request to an e-commerce site. The top axes show the web server, the bottom the database server. Threads which execute on behalf of this transaction are shown in the paler colour when blocked, and a darker colour when running. Packets sent and received are shown on the 'Net TX' and 'Net RX' lines respectively.

of web interactions processed per second by a simulated online bookstore (within the strictly specified parameters of the benchmark)[7]. Benchmarking stresses the system using some predetermined set of operations, and hence obtains the average values of performance counters for the resources of interest. To give a concrete example of applying the benchmarking approach, the workload of a particular request, say 'AddToCart', is determined by running AddToCart repeatedly and monitoring the demands on CPU, disk and network.

Benchmarking techniques are an attractive mechanism for measuring resource usage, as they are straightforward to perform, and experience of gathering workload statistics for Indy has shown this approach to give reasonable results. When the benchmarked system is subsequently simulated inside Indy its throughput generally comes within 5% of that observed on the real system while carrying out the benchmarking. However this approach is not ideal because it not only takes human effort, but also inevitably involves some averaging since is not possible to account resource usage at each component in the system to the single transaction which is responsible for that usage. In addition, the use of benchmarking to obtain usage statistics for performance modelling is critically flawed in two ways:

- In order to attribute the observed resource usage to a particular operation, that operation must be run in isolation, with the result that the pattern of resource usage can be skewed. It is very difficult to accurately model the behaviour of concurrently shared resources without implementing the actual system as shared resources can have a marked impact on performance. For example poor caching performance can significantly affect the system throughput, and chances are a single transaction run over and over again

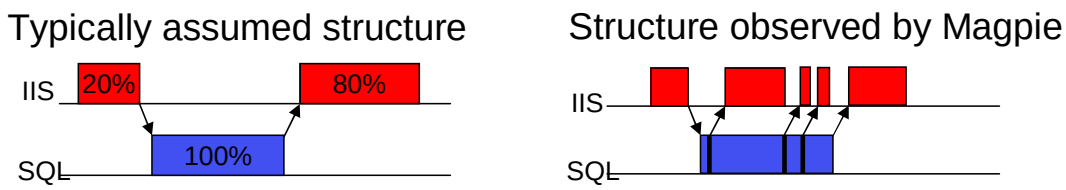

Figure 2:

will get a much better cache hit ratio than is ever realistic.

- The use of long-term averages to determine typical resource utilisation hides low level behaviour which may be significant, for example, network latency.

The latter point is fairly subtle, but can have significant consequences for performance modelling. When constructing a representative workload, the performance modeller measures the long term amount of resource, such as CPU cycles or disk access, consumed on the servers in the system. They must also make a guess at the structure of each type of transaction, that is the communication patterns and the degree of parallelism between separate components. Typically the structure is assumed to be a simple model as depicted in left-hand image of Figure 2. This represents a reasonable approximation of the structure of a request to a web server which makes a database access. The assumption is that the web server processes the request, does an RPC to the SQL server and when the results come back they are rendered as HTML and sent back to the client. However in reality, communication patterns are often more complicated with overlapping, out-of-order request processing, pipelining of data returned in a response, and so on. The right-hand diagram of Figure 2 shows the structure which we actually observe on the e-commerce system using Magpie. Performance simulations using these two different structures for identical resource usage give throughput and utilisation results which vary by up to a factor of two[8]. This misunderstanding of the behaviour of a transaction makes performance profiling and debugging even more inaccurate and difficult. The Magpie infrastructure is intended to improve the accuracy of workload models by extracting a fine-grained characterisation of each transaction type under *realistic* load conditions.

We anticipate that similarly complex transaction models will occur in the context of Web Services. In addition, experience in the past with GUI programming has shown that event-based concurrency is extremely challenging for programmers to get right. RPC in the wide area is a disaster waiting to happen both in terms of performance and in terms of ensuring correct behaviour. Support for the Magpie data-driven approach to performance profiling in the wide area through a standardized performance API is essential to address these problems.

4 Conclusion

There is a compelling case for applying the Magpie technique to the loosely-coupled distributed system which comprises the Web Services environment. As argued above, experience with a tightly-coupled distributed system has shown that high-level system counters are insufficient for understanding the performance of individual transactions, yet from the point of view of the Web Services user, a transaction-oriented view of performance is essential. In ad-

dition to performance insights, we believe modelling at this level can help with system design. Future research is considering how a performance API for Web Services might be defined, notwithstanding other important concerns such as security, timeliness and consistency.

References

[1] World Wide Web Consortium. Web Services. `http://www.w3c.org/2002/ws/`, 2002.

[2] Google. Google Web APIs. `http://www.google.com/apis/`, 2002.

[3] Microsoft. Microsoft .NET Passport. `http://www.passport.com/`, 2002.

[4] Jeffrey Dean, James E. Hicks, Carl A. Waldspurger, William E. Weihl, and George Chrysos. ProfileMe: Hardware support for instruction-level profiling on out-of-order processors. In *Proceedings of Micro-30*, December 1997.

[5] Galen Hunt and Doug Brubacher. Detours: Binary interception of Win32 functions. In *Proceedings of the 3rd USENIX Windows NT Symposium*, July 1999.

[6] Jonathan C. Hardwick, Efstathios Papaefsthathiou, and David Guimbellot. Modeling the performance of e-commerce sites. In *Computer Measurement Group 2001 Conference*, December 2001.

[7] Transaction Processing Performance Council. *TPC Benchmark W (Web Commerce) Specification.* Available from `http://www.tpc.org/tpcw/`.

[8] Rebecca Isaacs, Paul Barham, and Richard Mortier. Magpie: Data-driven performance analysis for distributed systems. Submitted for publication, September 2002.