REAL-TIME PERFORMANCE MONITORING FOR BUSINESS-CRITICAL APPLICATIONS

The Technological Advantages of Message Logging

Diamond Sierra
Enhancing the Quality of Information Technology Worldwide

PART I – MESSAGE LOGGING

PART II – COMPARATIVE ANALYSIS

INTRODUCTION

Performance monitoring of e-business applications is a critical issue facing the IT community today. As these applications manage an increasing number of transactions with both vendors as well as customers, their reliability is crucial to an enterprise's profitability. Achieving reliable performance is arguably even more important than expanding functionality and feature sets. Studies repeatedly demonstrate that customers will abandon a slow web application before they ever have a chance to be dazzled by its features. Furthermore, corporate partners relying on integrated B2B applications will be unimpressed with robustness of functionality when guaranteed Service Level Agreements are not realized.

Just how critical is application performance? Studies from business research firms such as Forrester, Jupiter, and Zona reveal that an e-commerce site has a mere eight seconds to respond before an average customer abandons a site. In fact, nearly half of all users have stopped using a preferred site because of performance difficulties. Moreover, severe performance degradations typically cause transaction timeouts, resulting in purchase failures. Studies show that 80% of online consumers have experienced a purchase failure and that 25% of those will *never* return to that site. Furthermore, that same research shows those customers will tell an average of ten other people about their bad experience. With abandoned purchases estimated at twenty-five cents for every dollar of revenue, the negative impact of bad performance on sales as well as corporate image can clearly be a company's most significant bottom line factor.

There are numerous Application Management Systems (AMS) such as Tivoli, CoManage, BMC Patrol, and NetIQ that offer various performance monitoring products. These products, however, are based on technologies that have a number of shortcomings in relation to real time performance monitoring in the custom application environment. This paper discusses the limitations of the traditional AMS performance monitoring solutions and the means by which these shortcomings are overcome through the use of message logging technology.

DEFICIENCIES OF TRADITIONAL SOLUTIONS

The standard offerings for performance monitoring largely consist of two approaches: 1) continuously monitoring operating system, database, and network performance related metrics and 2) continuously running simulations through the system and evaluating them. Both typically operate in conjunction with sophisticated analysis, reporting, and alert engines but in the final analysis these may reveal little about the quality of the customer's experience.

System metrics and simulations often fall short because they only provide *inference* information about the real issue – the speed of actual customer transactions. CPU utilization, memory usage, free disk space, database activity, and network traffic all measure up well within tolerance when the application is simply hung and not processing any transactions at all. Similarly, timings of test simulations appear fine when they omit the database operations by accessing only locally cached data sets created by identical data requests from identical simulations.

Misleading data, caching facilities, and other defeating mechanisms cause performance monitoring products based on system metrics or simulations to fail to recognize a performance shortfall. What is needed instead is simple real-time monitoring of actual live transactions. The sections below discuss precisely how to use message logging technology to easily and effectively achieve this.

THE MESSAGE LOGGING SOLUTION

To fully appreciate the completeness of the message logging approach to performance monitoring, it is helpful to understand its comparative advantages. The fundamental strength of message logging is that it enables real-time monitoring of *actual transactions* – each and every one of them. This realizes a number of distinct and decisive advantages over other technologies that track only simulation times and system metrics:

1) Enables monitoring of actual live transactions/operations – not potentially misleading simulation times and environment metrics
2) Enables monitoring of sub-transactions to provide advance warning before problems manifest visibly at higher levels
3) Enables monitoring of events in custom developed applications – not just off-the-shelf applications with standard public interfaces for AMS products
4) Operates seamlessly in distributed systems spanning components, machines, networks, programming languages, and operating systems. Also operates independent of browser interfaces typically required by AMSs utilizing simulation approaches – thus lending itself to environments such as process control, internal business systems, and B2B applications.

So what *is* message logging?

Prior to discussing the means by which message logging technology delivers the advantages listed above, it is important to define the term. Many in the IT community have either not worked with the technology previously or have worked only with lightweight utilities rather than production quality tools such as Diamond Sierra's Paul Bunyan. (Named after the legendary logger – Paul Bunyan.) Basic message logging tools, like the Windows SDK's DBWin, are simple yet useful utilities. They allow developers to 'log' error, debugging, and general status information from within application code such as below:

```
. . .

LogMsg ("Entering MySubRoutine: Customer name = %s", szCustomerName);

if (Length(szCustomerName) == 0)
   {
   LogMsg ("Error: Customer name parameter cannot be zero length.");
   return;
   }

. . .

LogMsg ("Leaving MySubRoutine");
```

More advanced professional grade message logging systems like Paul Bunyan or the UNIX/Linux SysLog facility allow for (and often automate) inclusion of additional contextual variables such as severity, source code filenames and line numbers, process and module names, and obviously timestamps, and not so obviously but (as will be shown below) equally important, application defined transaction ids. These tools also provide the ability to consolidate log messages in live production environments across distributed systems spanning components, processes, machines, and even the Internet. Sample logging output revealing internal timing information is shown in the following screen capture:

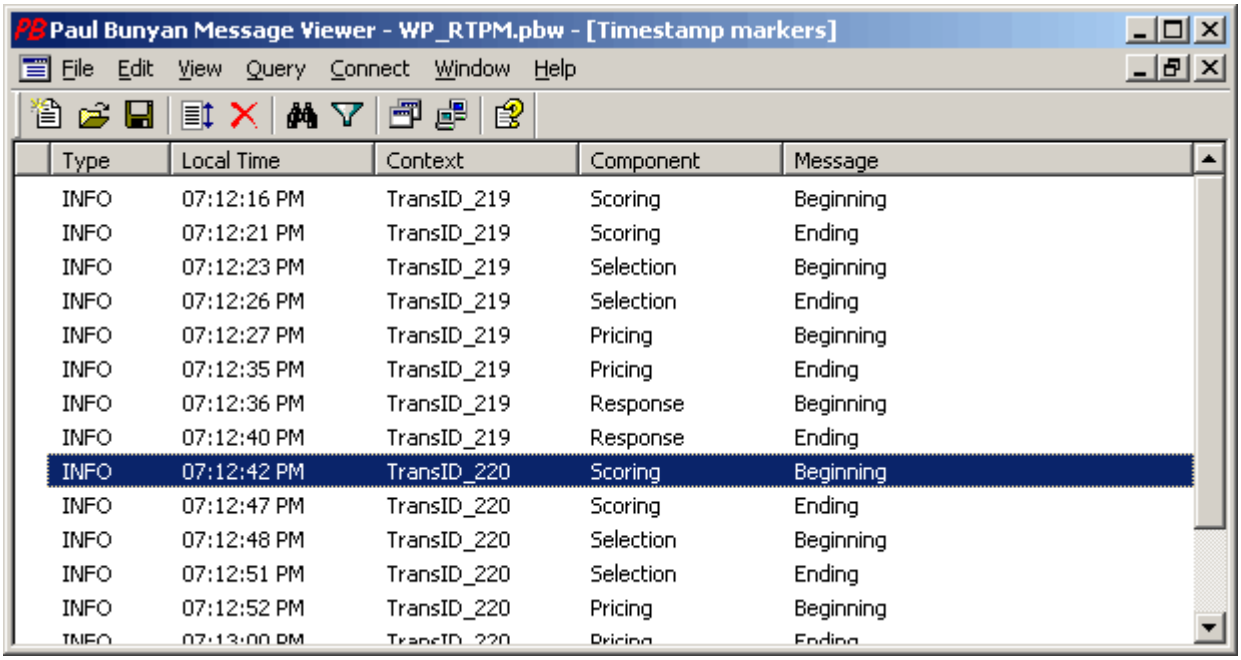

Figure 1

… and how does message logging enable real-time performance monitoring?

Using event logging to evaluate performance is a straightforward process. The timestamp information contained in the log messages is used to determine the beginning and ending times of the various events of interest. This timing information is then (note) grouped by transaction id and separated by component. In this form, the timing data in the log messages enables complete and detailed analyses of an application's performance, to whatever degree of granularity desired. For example, the following chart illustrates a full performance profile of a typical multi-component, parallel processing application such as an online loan approval system – broken down by individual component *and* by individual transaction:

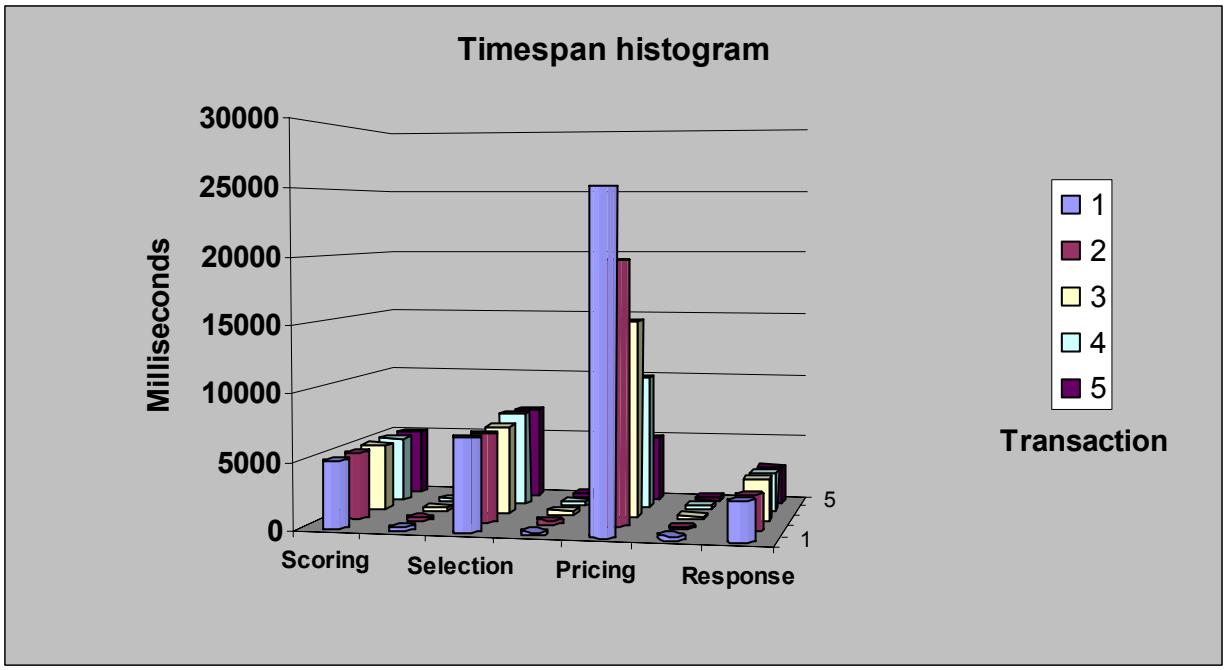

Figure 2

The above chart is of course only a static snapshot created in a post-run situation after processing was completed. It does, however, serve to illustrate the sheer degree of performance evaluation capable using only the data provided by the logging. While very useful for static performance analysis, the logical extension – dynamic performance *monitoring* – is to route the log messages to processing entities that programmatically evaluate each operation's timing data as it is logged, carrying out such performance evaluations in real-time.

Note that the utility of real-time performance evaluation is not to dynamically update performance indicators on remote command consoles. Rather, this functionality is used to implement the business critical tasks of automatically generating alert notifications to support staff (via email, pagers) *and* automatically taking corrective action to resolve the problem programmatically (archiving/purging database tables, spawning redundant services to distribute workload). Automation of these tasks serves to alleviate many performance issues before they become apparent to the user, provides advance warning of impending problems to support staff, and provides instantaneous alert notification when such issues do arise, thereby reducing response and resolution times. Frequently, these enhanced efficiencies not only increase application performance and reliability and ultimately profitability but simultaneously reduce support and maintenance expenses as well.

The utility and value of message logging based performance monitoring is evidenced in the numerous companies that have custom built powerful logging systems to gain just such functionality and benefits. Many others have used operating system logging facilities such as SysLog for UNIX/Linux or have purchased off-the-shelf commercial grade logging and performance monitoring systems such as Diamond Sierra's Windows based product line.

ADVANTAGES OF MESSAGE LOGGING TECHNOLOGY OVER OTHER METHODS – A DETAILED EXAMINATION

Having illustrated the message logging approach to real-time performance monitoring, it is useful to revisit the comparative analysis to other technologies in greater detail.

Message logging monitors actual live transactions – not interspersed simulations

The technique of measuring the timing of simulations to infer the timing of live transactions has two primary shortcomings:

Performance of simulations and live transactions varies greatly

The first major shortcoming is that the underlying assumption that simulated transactions take on average the same amount of time to process as live transactions is often inaccurate. Many applications are of sufficient complexity that the processing path through them, and subsequently the time taken to process, is very much dependent upon the data itself. For example, a customer's age, number of applicants, dollar amounts, tax regions, availability of inventory, or even missing or erroneous data fields can greatly affect the processing path. Were it even possible to create a test suite to monitor the majority of combinations of elements/paths with each new code release, it would be counterproductive to regularly tax live production servers with such a barrage of tests.

Performance enhancing cache facilities also factor in. The nature and in fact the very purpose of caching is to greatly increase the speed of repeated identical operations. Such systems are used pervasively throughout any large-scale application – in the web server, the database server as well as drivers, memory caching within the operating system, and resource pooling of connection objects, threads, and a host of other entities. These systems are specifically designed to react to repeated operations (e.g. repeated simulations) by actively tuning themselves to execute them faster on subsequent identical runs. Regular repeated simulations will by their very nature execute at speeds that increasingly differ from those of unique live transactions.

Randomly sampled events means randomly missed events

The second major shortcoming with the simulation monitoring approach is that poorly performing transactions have a high probability of simply going unnoticed. Even assuming the highly unlikely situation where simulations and live transactions perform similarly, performance anomalies are nonetheless often created by low level mechanisms that ultimately cause them to be distributed essentially randomly across all transactions, simulated or live. The problem this creates is simply one of statistical sampling: when sampling a random apple from a bag containing nine good and one sour, there is a ninety percent chance that the sour apple will go undetected. If the purpose of the test was to determine if there were any sour apples in the bag then the test is clearly inadequate.

In terms of performance monitoring, consider a thousand customer per hour e-commerce web site monitored with timed simulations carried out continuously at ten minute intervals. A performance problem that randomly causes half of the application's transactions to time out has a 25% chance of passing two tests in a row without detection – resulting in an ongoing performance problem affecting fully *half* of the *five hundred* customers during that initial thirty minute period. And those are just the customers affected during the time taken to *detect* the problem; not to *correct* it.

The solution in the above example might seem to be to simply increase the simulation/sampling rate but even increasing it ten-fold still yields inadequate results. Even in an application sampled once every minute, the above performance problem will still impact *dozens* of customers *one out of every four* times it occurs – even with a simulation frequency now above 5%. Clearly, the point is quickly reached where the simulation load itself is degrading the performance of the production servers, with the problem still unresolved. This is on top of the fact that the real performance is degraded further still by the fact that the sheer volume of simulations is artificially skewing, and consequently defeating, all of the performance

enhancing caching systems used throughout the application. And, again, this is all predicated on the very unlikely assumption that the performance of the simulations actually reflects that of the live transactions.

The solution: discard simulations and measure the actual transaction times directly
Message logging offers the perfect 'elegance in simplicity' solution: abandon the use of simulations entirely and simply monitor the live transactions themselves – each and every one of them. Measuring the live transactions directly makes moot all issues associated with time differences between simulated and actual activity, regardless of underlying cause. At the same time it also fully eliminates all questions of even one sour apple slipping by unnoticed. Further, the message logging approach achieves these positive results while simultaneously eliminating both the additional production server loads *and* the adverse cache system impacts incurred by the use of simulations.

Message logging measures performance directly – not by indirect inference from operating system, database, or network metrics

The assertion that application performance can be inferred by analysis of underlying subsystem metrics is simply not true. This is demonstrated in the following two case studies, both of which occurred in actual large scale production environments:

Case 1:
Two connectivity servers at respective ends of a B2B integration communicate by method of Server A regularly polling Server B to inquire for incoming transactions. Upon poll, Server A waits for a response from Server B and upon receiving it then forks between processing any new transactions or simply polling again later. Due to a bug in the underlying transport, Server B intermittently never receives Server A's inquiry and so does not respond. Server A meanwhile goes to sleep waiting for Server B's response that will never come. The servers are now out of sync and the communication effectively suspended, and the entire system (a multimillion dollar B2B application) is effectively shut down just as completely as if the power was off.

Case 2:
A distributed e-commerce application has one of its COM components configured incorrectly with the result that the code is executed by the operating system in single threaded fashion even though the component was correctly written to operate multithreaded. The outward manifestation is that the component processes requests not in parallel, and not even in FIFO order, but in *LIFO* (Last In First Out) queue fashion. As long as transaction requests come in essentially one at a time there is no performance issue but if a hundred come in rapidly enough to overlap then the last request finishes in expected time but the first request, preempted ninety-nine times, takes over an hour.

The above two examples are not hypothetical scenarios but true occurrences reported from Diamond Sierra customers with real production systems and with substantial AMS installations monitoring the whole environment. Both cases originally went undetected by the extensive AMS implementations in place which were only able to monitor system metrics – not the actual transaction times.

In Case 1, the polling transaction (the request/response) that normally takes a few seconds would extend to hours, being 'discovered' typically by customers' inquiries to the help desk. The timing failure was not, and *could* not, be detected by the AMS with all its metric analysis because the monitored subsystems were all working fine – operating system, database, network, etc. – Server A was properly waiting for a response that Server B didn't know it was supposed to send. The problem was remedied by simply logging a "Sending request…" message followed by a "Received response" message and using Diamond Sierra's performance monitoring products to programmatically process the two and issue an alert if the second message was more than ten seconds overdue.

In Case 2 as well, with the misconfigured COM component, the resident AMS did not detect any performance problem because all of the various subsystems were again operating normally; they were simply processing requests in an unintended order. Case 2, however, illustrates yet another fundamental

problem with the metric approach which is that AMSs support no representation of the concept of high level transactions and thus cannot tell one transaction time from another – only that the subsystems those operations utilize are operating as designed. While this is a special case, it clearly demonstrates again the general deficiency of assuming that low level metrics are indicative of high level operations. This class of performance problem as well is easily monitored and detected using the same techniques used to resolve the issues in Case 1.

It should be noted that in addition to fully resolving the above AMS performance *monitoring* deficiencies, the problems in both Case 1 and Case 2 were also initially *discovered and identified* through analysis of the timing information within the message logging. For example, analyzing the transaction and component data from the previous chart (Figure 2) on an absolute time scale plainly reveals exactly the preemptive LIFO processing behavior of Case 2. See Figure 3 below.

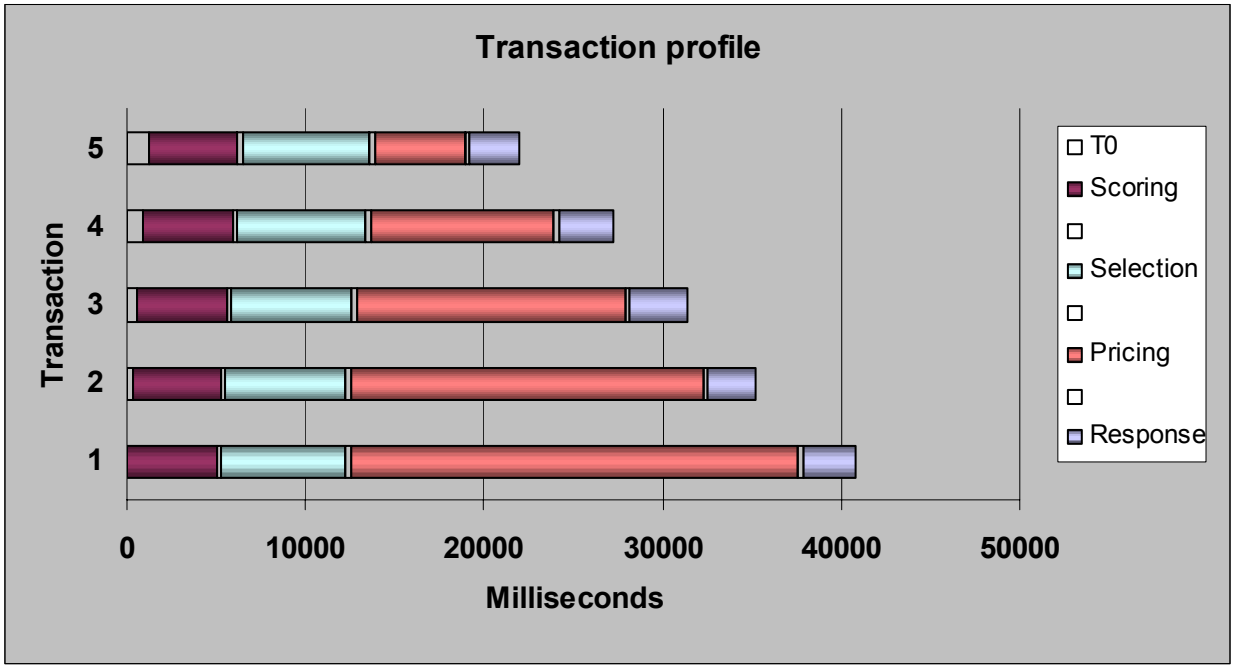

Figure 3

Message logging based performance monitoring is a broader and more flexible solution than that of monitoring simulations or system metrics

Supports information consolidation across widely distributed environments
Advanced logging systems like SysLog and Paul Bunyan support collection and consolidation of log messages over any TCP/IP connection. As such, it is a simple matter to leverage information from all components of a distributed application regardless of location, even components spanning Internet connections. These remote data consolidation capabilities can be applied not only to distributed systems spanning multiple computers but also to transactions spanning multiple endpoints. Many applications support classes of operations that begin on one server and terminate on another or even several others – a common scenario in many e-commerce and process control systems.

Enables the tracking of operations by use of application defined transaction ids
The use of application defined transaction ids is critically important as it allows subsequent performance monitoring operations to analyze and evaluate not just atomic operations at the subsystem level but to

identify and track the individual composite transactions themselves. It should be noted that the performance issues of Case 2 above were both discovered and resolved as a direct result of the ability to separate the timing information by transaction id – an ability provided by message logging exclusively.

Additionally, since transaction ids are provided via the logging itself, tracking of transactions and associated timing information is entirely independent of any required native support from the various transports and operating systems a transaction may traverse. That is, a transaction could easily begin as an email on a Linux server and end as an update in a database stored procedure on a Windows server elsewhere on the Internet yet be tracked and timed seamlessly across all points..

Eliminates the requirement of application support for unnecessary public interfaces
Another advantage of the message logging approach is that it is completely independent of the type of application. For example, virtually all simulation based techniques rely on the application's support of a web interface, which an AMS can then use to simulate and time browser based transactions. However, many large scale applications such as B2B and process control or internal data processing systems have no web interface. Message logging has no such limitation though – start/stop information is easily logged and collected from any component or application independent of communication interfaces.

Enables precise monitoring of internal sub-operations
Message logging also supports the monitoring of timed events down to any desired granularity – the degree and granularity of monitoring information is entirely at the discretion of the IT staff. In the above case study of the B2B connectivity servers' communication failure, the operation timed with begin/end log messages was literally the execution of a single API call – a single operation which at times simply 'hung'.

Enables advance warnings of impending performance issues
Lastly, fine granularity of event timing is also useful in providing advanced warning of poor performance conditions. Consider a multi-component system. While timing the whole transaction as a unit provides the information necessary for alert, the condition can often be detected significantly earlier by monitoring at the level of the individual components. Alerts are then issued immediately when any particular component begins to operate out of tolerance instead of later when the problem has worsened enough to push the entire transaction time out of tolerance. The advance notice of the performance condition provides support staff with additional time to address the problem, often enabling them to avert performance issues at the transaction level before they occur.

SUMMARY

IT companies spend millions, often tens of millions, of dollars developing custom software applications that are the very foundation of their financial bottom line. Studies have repeatedly shown that the subsequent profits realized from these applications rests substantially on the reliability of their performance – whether they are e-commerce applications evaluated on web site response time or B2B applications monitored for compliance with Service Level Agreements.

In the performance monitoring of custom software applications, the overwhelming advantage of message logging based solutions above all other technologies is the critical ability to utilize internal application-specific information. It is this information which uniquely enables the real-time monitoring of actual transaction times, down to individual application components. It is only the monitoring of actual transaction times that avoids the pitfalls of misleading information from system metrics and simulations and provides the accuracy of information necessary to ensure reliable application performance.