Visualising the performance of a large transaction processing computer system

by Rupert Brown

Being able to monitor and assess the performance of a large transaction processing system is an essential activity during the commissioning and acceptance phases of the system's development and then subsequently when the system enters operational service. This article describes some of the data capture and presentation techniques used on a major new road tolling system in the Southern Hemisphere.

Modern road tolling systems seek to minimise the inconvenience of using the toll road. Conventional toll roads typically require a motorist to stop their car at a toll booth and have ready the toll amount in cash. Modern systems, based on electronic tag technology (Fig. 1), allow a road user to open an account with the road operator and have tolls automatically charged to their account. The need to stop at a booth and have cash available is removed, facilitating a smoother, faster journey.

A toll account holder fixes an electronic tag in his or her vehicle. Tolling gantries are placed at designated points along the road. As the road user drives under a gantry, the gantry is able to interrogate the tag and determine its unique identification number. This number, along with date and time information, and the number of the gantry, is wrapped into a message, which is sent back to a central tolling computer. The central tolling computer is able to unwrap the message and combine it with data from other gantries to reconstruct the trip driven by the road user. A toll is then levied against that user's account.

At any given time the road may be in use by thousands of vehicles. As a result, the central toll computer is being sent many thousands of messages an hour, all of which have to be processed quickly. The system is not strictly real-time, but the criterion is that, within a defined time period, such as 24 hours, the system will have been able to process all the messages it has received within that period, so that no message backlog exists at the end of the period.

In this situation it is vital that the processing performance of the central tolling computer is sufficient to cope with the anticipated vehicle load before the road is opened. After the road has been opened, there remains a need to ensure that the system is able to handle the actual vehicle load.

The goal and the task

The main goal is twofold:

- To get into service a processing system that has been shown to be capable of handling the anticipated load in tests using simulations.
- To demonstrate that, when in service, the system can handle the actual load and, given that vehicle load increases, to be able to predict when the system will reach its processing limit.

In order to achieve the goal the performance visualisation task becomes:

- To understand the anticipated load and find a way of translating the anticipated load into performance requirements.

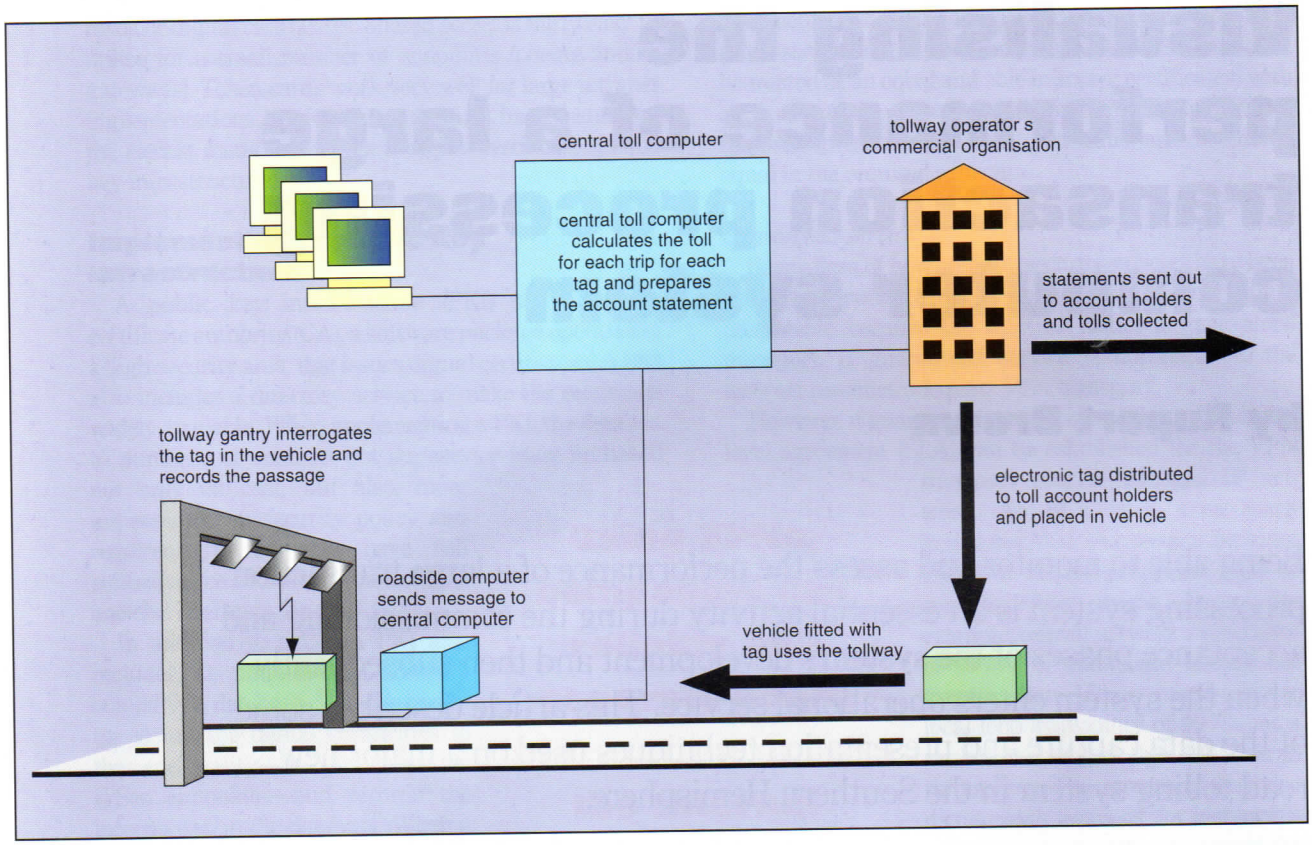

Fig. 1 Overview of the electronic tag toll road system

- To take those performance requirements and relate them to real things that can be measured within the context of the processing system.
- To capture metrics data.
- To format that data in such a way that it serves the needs of the project's stakeholder groups.

Performance targets

Anticipated load should be a product of the project definition phase, or some other early project study. Typically a figure for anticipated load will be stated in the context of real world entities, be it telephone calls per minute, bank transactions per hour, or in this case vehicles using the road. Getting the absolute values correct is obviously a crucial task, but as important is being exact about the definition of the units in which those figures are expressed.

To use a contrived example, suppose the builder of the toll road lets two separate contracts, one for the gantry equipment and one for the central tolling computer. Both contracts contain the requirement that the systems should be able to process half a million vehicles passing under the gantries each day. Both contractors think their systems will be capable of meeting this level of performance. But then after some design work the gantry contractor realises that, in order to service other requirements, it will actually have to send the central toll computer two messages for every vehicle that passes under a gantry. Whereas before the central toll computer contractor was looking at a performance requirement of 500 000 messages a day, in one go that has risen to one million.

Then, as the project progresses, it becomes clear that what was meant by the original requirement was not '500 000 vehicles therefore 500 000 messages', but '500 000 vehicles each of which could travel under several gantries'. The actual message load goes up again.

So thinking about performance requirements in terms of real-world entities is only the first step. Performance requirements, at some stage and the sooner the better, need to be expressed in terms that are directly relevant to the design of the system. From the example given above, the starting point is vehicles per day, but that becomes messages per day within the context of the system, and the two numbers are obviously not the same.

Big projects last several years, and in that time things change. It is vital that, when a change occurs to a performance requirement, both the absolute value of the requirement and its units are considered. Always bear in mind that, once certain aspects of a design have been agreed and implemented, a performance requirement change can mean throwing the system away and starting again.

Measuring

There is a useful acronym to keep in mind when writing

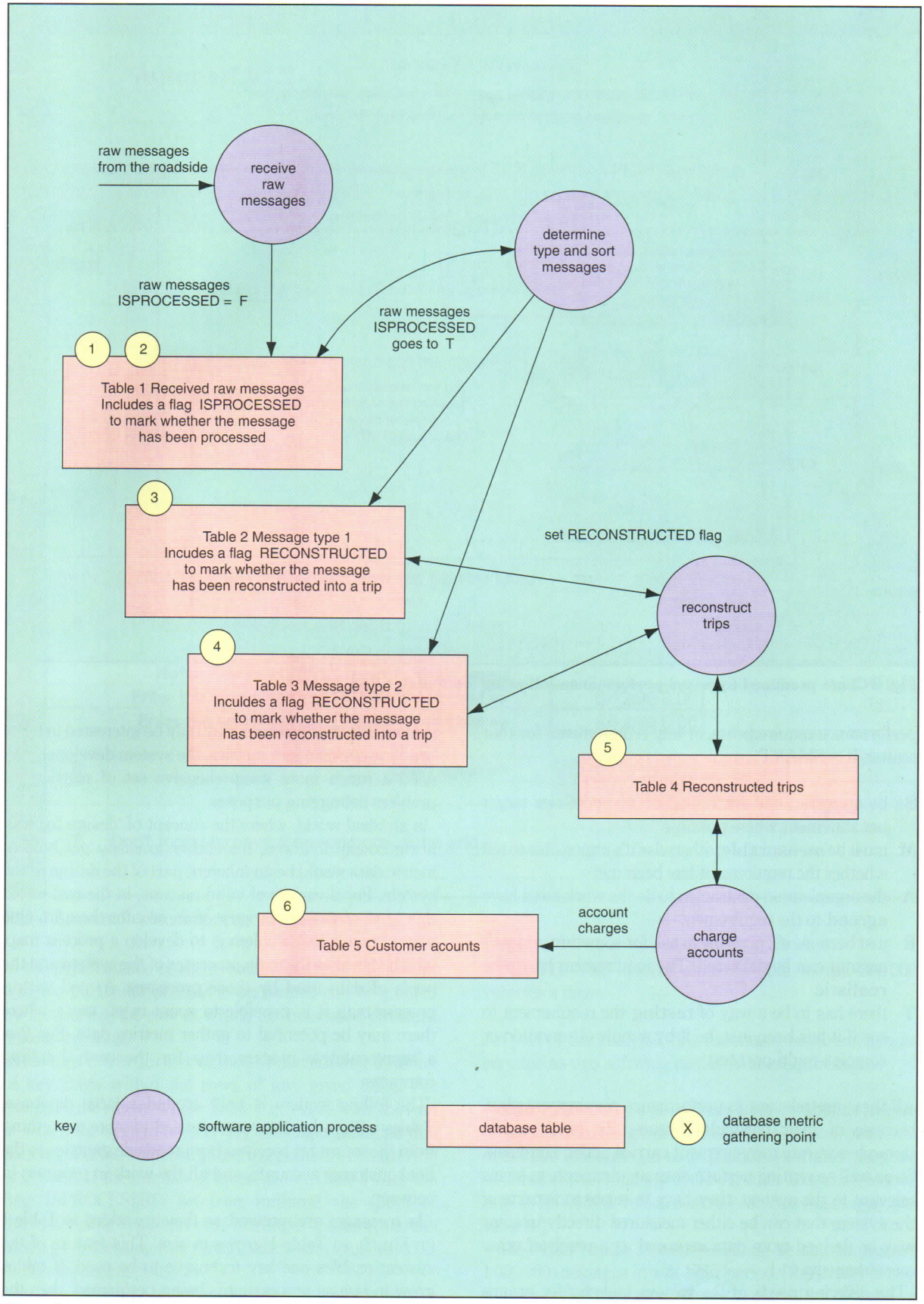

Fig. 2 Central toll computer—representative process map with metric gathering points

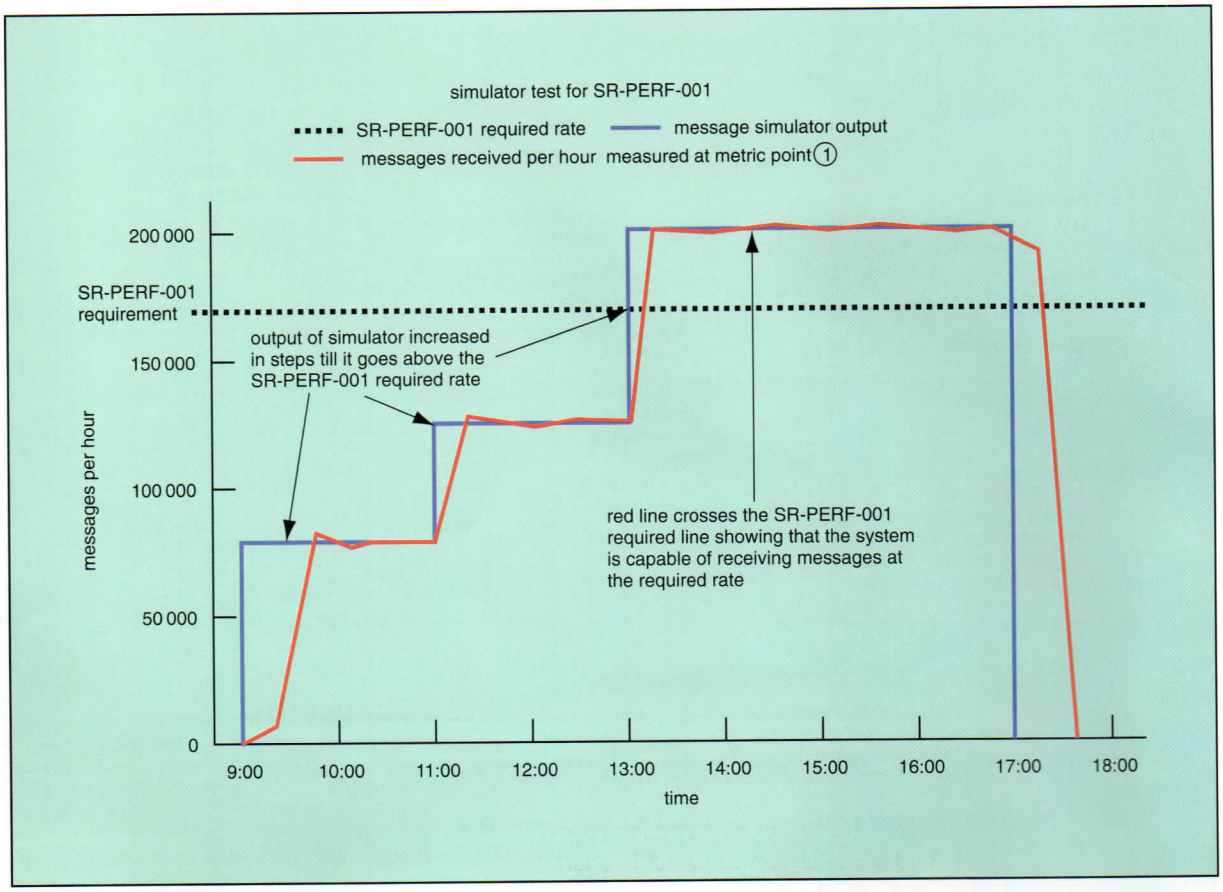

Fig. 3 Chart produced to reveal performance following a simulator test

performance requirements, or any requirements for that matter. It is SMART:

- **S** be **specific**, one attribute, one function, one target per statement where possible
- **M** must be **measurable**, otherwise it's impossible to tell whether the requirement has been met
- **A** the organisation contracted to do the work must have **agreed** to the requirement
- **R** just because it's possible to ask for something doesn't mean it can be delivered. The requirement has to be **realistic**
- **T** there has to be a way of **testing** the requirement to see if it has been met, be it by simple observation or complex multi-part test

All these are relevant to performance requirements but, because of their particular nature, the need to test through accurate measurement carries added emphasis. So as well as writing performance requirements in terms relevant to the system, they have to relate to aspects of the system that can be either measured directly in some way or derived from data captured as a result of other measurements.

The differing needs of the various stakeholder groups also influence what should be measured. Whereas the system owner or procurer may only be interested in a few headline performance metrics, the system developer may need a much more comprehensive set of metrics for problem debugging purposes.

In an ideal world, where the concept of 'design for test' is rigorously followed, the ability to easily get hold of metric data would be an inherent part of the design of the system. For a variety of valid reasons, in the real world this kind of activity is very often an afterthought. One way to tackle the problem is to develop a process map, which identifies the main processes of the system and the pools of data used by those processes. Armed with a process map it is possible to zoom in on areas where there may be potential to gather metrics data. Fig. 2 is a representative process map for the central tolling computer.

The tolling system is built around a large database. Tables within the database are used to store everything from the messages received from roadside gantries to the final customer accounts, and all the work in progress in between.

As messages are received, so they are added to Table 1 (in Fig. 2), so Table 1 grows in size. This feature of the system enables one key technique to be used. If tables grow in size as processing is being performed, then the rate at which the tables grow is a very good indicator as

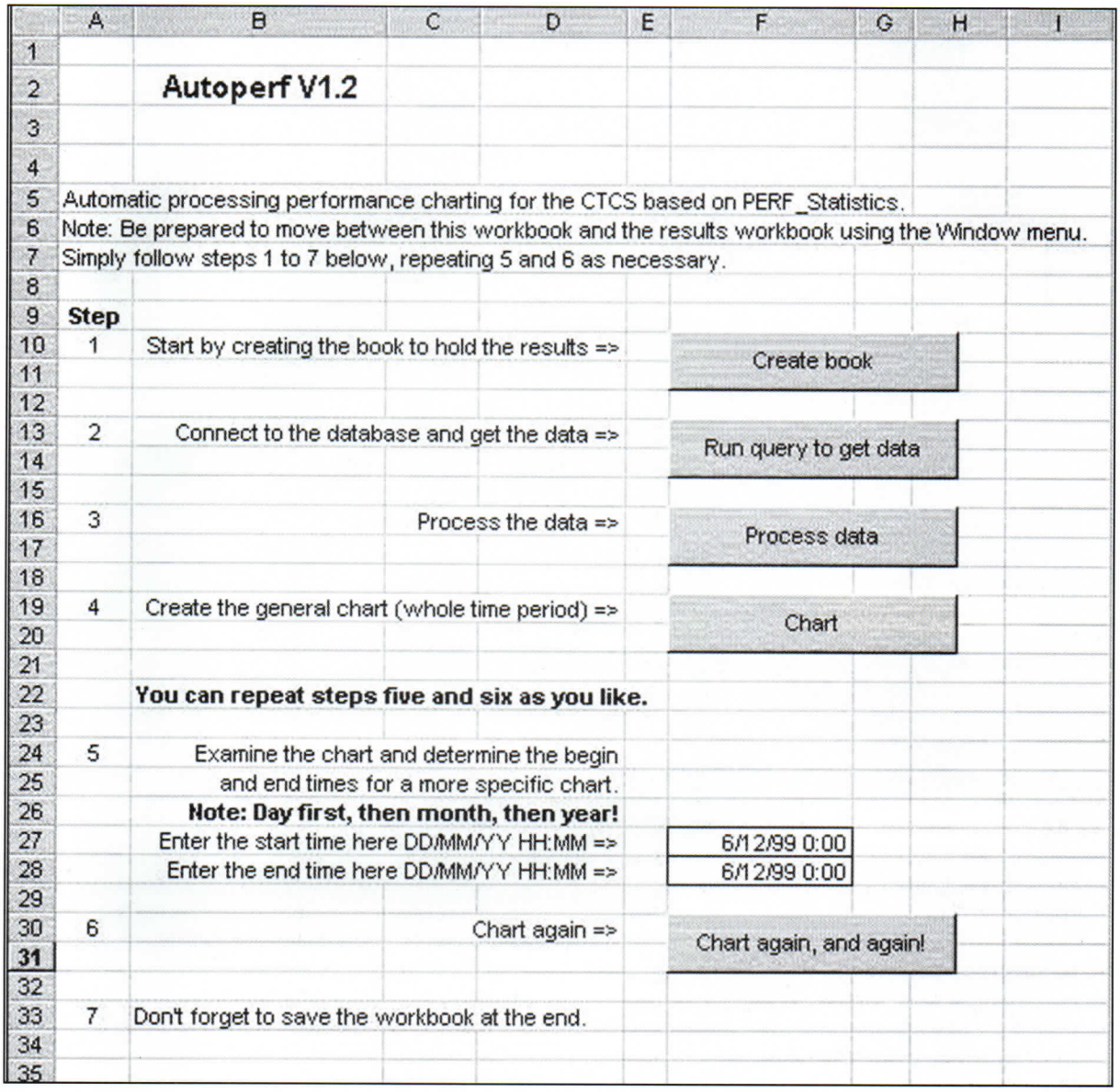

Fig. 4 Automatic charting program user interface

to the performance of the system. Generally the rate at which a particular process is performed can be gauged by measuring the rate at which its associated database table is growing.

This idea can be further refined by considering the state of key flags within the rows of any given table. For example, the rate at which Table 1 grows reveals the rate at which the system is receiving messages (imagine this data is collected at metric gathering point ①). But the rate of growth of the number of messages in Table 1 with the flag 'ISPROCESSED' set true, indicates the speed of the 'determine type and sort messages' process, i.e. the message processing rate (imagine this data is collected at metric gathering point ②).

As an example, assume that the following was a functional requirement of the system:

- **FR001:** The system shall be able to toll 500 000 vehicles a day.

As part of the analysis, this functional requirement was restated as two software performance requirements:

- **SR-PERF-001:** The system shall be able to receive messages from the gantries at a combined total, peak rate of 170 000 messages an hour. [Rationale: traffic flows are not uniform over a daily cycle. At rush hour the effective rate is 2 000 000 vehicles a day at two messages per vehicle.]
- **SR-PERF-002:** The system shall be able to process 1 000 000 messages in a 24 hour period. [Rationale: in any 24 hour period the system has to be capable of processing all the messages it receives.]

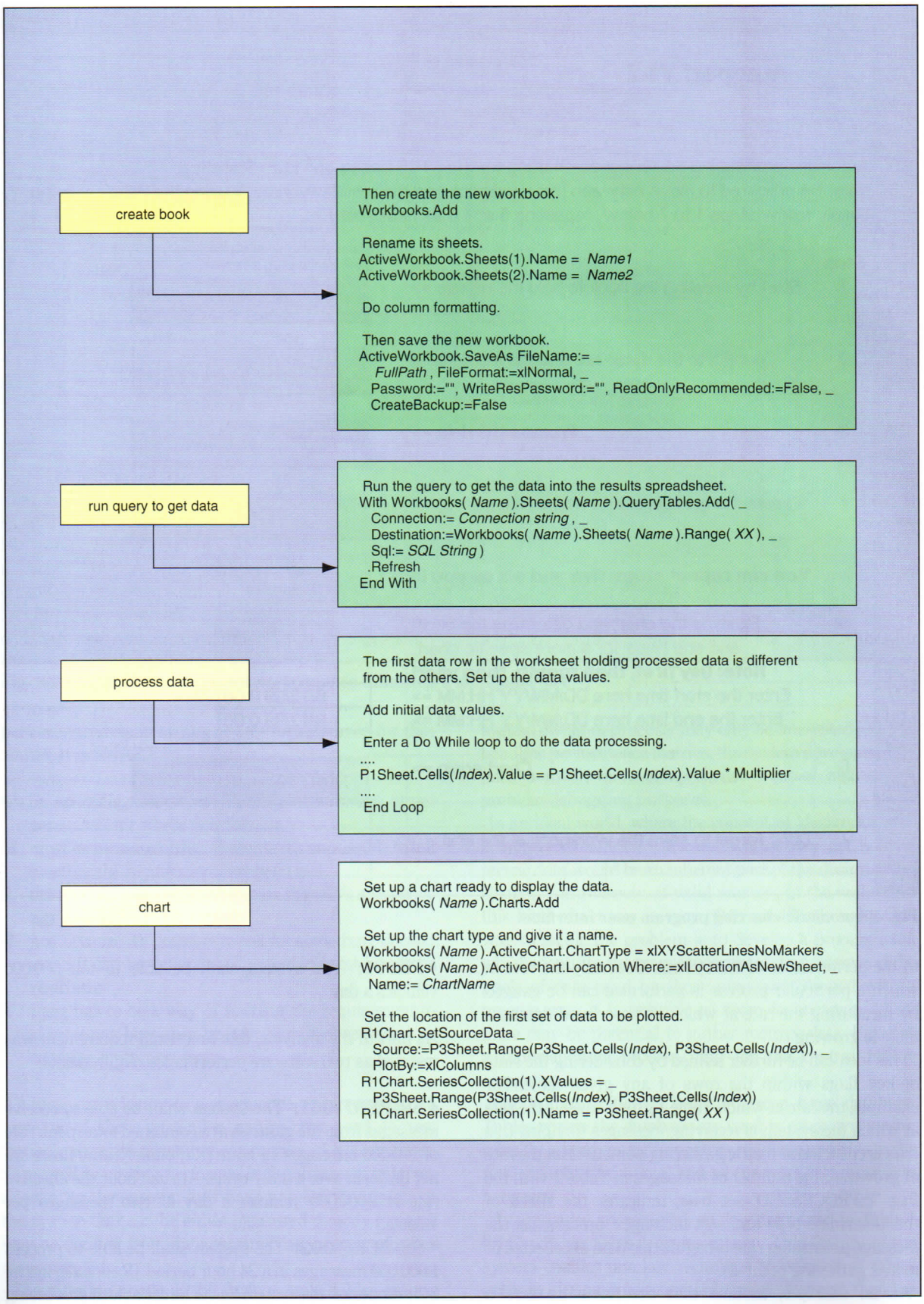

Fig. 5 Automatic charting program structure

It follows therefore that the data collected from metric gathering point ① will reveal whether the system meets the requirement SR-PERF-001. Similarly the data collected from metric gathering point ② will reveal whether the system meets the requirement SR-PERF-002. The other metric gathering points would be determined and used in similar ways.

In this particular system, messages are not removed from database tables until a dedicated archiving process is initiated. It is a simple matter to keep track of the number of messages removed to archive and so compensate for the rate 'glitch' occurring at that time.

Other systems will require different techniques to be used to successfully gather data. Typically, with bespoke systems there is no single uniform way of this doing this. Different industries over the years have developed their own preferred measurements, and there are long-standing performance measurement techniques especially in the area of database and UNIX tuning.[1] But a process map and a little lateral thinking will help when confronted with a novel system.

Another question is: 'How much computing resource is being used to deliver this application's performance?'

Another related question is: 'How much computing resource is being used to deliver this application's performance?'. Such a question is usually driven by a requirement stating that a certain percentage of the computing resource should remain unused to allow for growth. Computing resource can be measured in terms of a percentage. 100% would mean that all the available central processing unit (CPU) time is being used. 0% indicates that the CPU isn't doing work.

This metric adds an extra dimension to the performance measurement question. Consider the situation where a system meets its processing performance requirements but takes 100% CPU time to do it. Such a situation would be less than ideal because it allows no room for unforeseen problems and system growth.

So metric data collection is driven by the requirements, the form of the system is defined by a process map or its equivalent, and other needs such as information for debugging purposes.

Data capture

Knowing the performance targets from the requirements, and knowing where to get the data to test the system against those targets, the task now is to capture the data. When considering a system that processes large amounts of data on a continuous basis the following requirements for a data capture process emerge:

- The process should be able to run continuously without the need for manual intervention.
- To measure something is to change it—the process itself should have no or very little impact on the actual application.
- The data delivered should be in a format that is easy to work with using standard office software tools.
- The source of the data and the date and time it was collected should be clearly marked.

The central tolling computer application runs on UNIX machines and doing things at regular intervals is made possible because the operating system provides the `cron` utility. Instructions can be written into a file along with timing information. The `cron` utility runs the instructions in accordance with the timing information. In this case the instructions were in the form of an SQL statement that interrogated the size of various database tables. The data returned by the SQL statement was itself written into a table dedicated to storing this metric data.

Running the SQL statement consumes CPU time. If the tables are very large, and the sampling is very frequent, the act of collecting the performance data can itself have a detrimental impact on observed system performance. The regime adopted in this case, with test runs that lasted anywhere between three hours and three days, was to sample the size of the database tables every 20 minutes. This level of activity appeared to have a negligible impact on computing resources, but at the same time gave a reasonable level of visibility of system operations.

The SQL query included a full date and time stamp that marked returned data with a time that went down to the level of seconds. Initially the query extracted metrics data into a simple ASCII file that could be readily imported

Panel 1 UNIX performance measurement utilities

`sar`	**system activity reporter.** A utility that is used to provide the CPU usage metrics: %usr portion of time running in user mode (the application) %sys portion of time running in system mode %wio portion of time idle with some process waiting for block I/O %idle portion of time otherwise idle
`vmstat`	**virtual memory statistics.** vmstat reports statistics about process, virtual memory, disc, and CPU activity such as cache flushing statistics and the number of interrupts per device.
`isostat`	**I/O statistics.** isostat produces measures of throughput, utilisation, queue lengths, transaction rates and service time.
`uptime`	The uptime command prints out the load average. The load average is the sum of the run queue length and number of jobs currently running on CPUs.

into a PC spreadsheet application for subsequent analysis. Later the process was improved so that macros written for the spreadsheet application could perform the extraction and chart the data automatically. This is discussed below.

UNIX provides ready-to-use utilities for monitoring CPU usage, I/O throughput and the effective load on the system processors. Detailed information on these utilities is outside the scope of this article, but Panel 1 summarises four of the utilities most used in this case. All these utilities produce formatted ASCII output files that may be imported into a spreadsheet application.

All these utilities produce formatted ASCII output files that may be imported into a spreadsheet application

Data format, presentation and use

On the toll road project, the collected metric data was typically in the form of a series of numbers separated out into columns with the first column usually containing the date time information. Numerical information of this kind is useful for checking very specific things about the performance of the system, but it is difficult to get a 'feel' for how the system is doing from row upon row of numbers. Metric data visualisation using charts is one solution to this problem. For the day-to-day monitoring of the central toll computer performance three basic chart types were employed:

- Processing rate chart—rates calculated from the database table size data (Fig. 2 metric gathering points) and the time between database samples. These rates were plotted on the y axis against time on the x axis (see Fig. 6).
- CPU usage chart information from `sar` was plotted on the y axis against time on the x axis (see Fig. 7).
- Load chart—the information from `uptime` was plotted on the y axis against time on the x axis (see Fig. 7).

These charts formed the basic performance information used by the project procurer and future operator. The developer would look at these too, but would supplement them with more detailed information derived from application

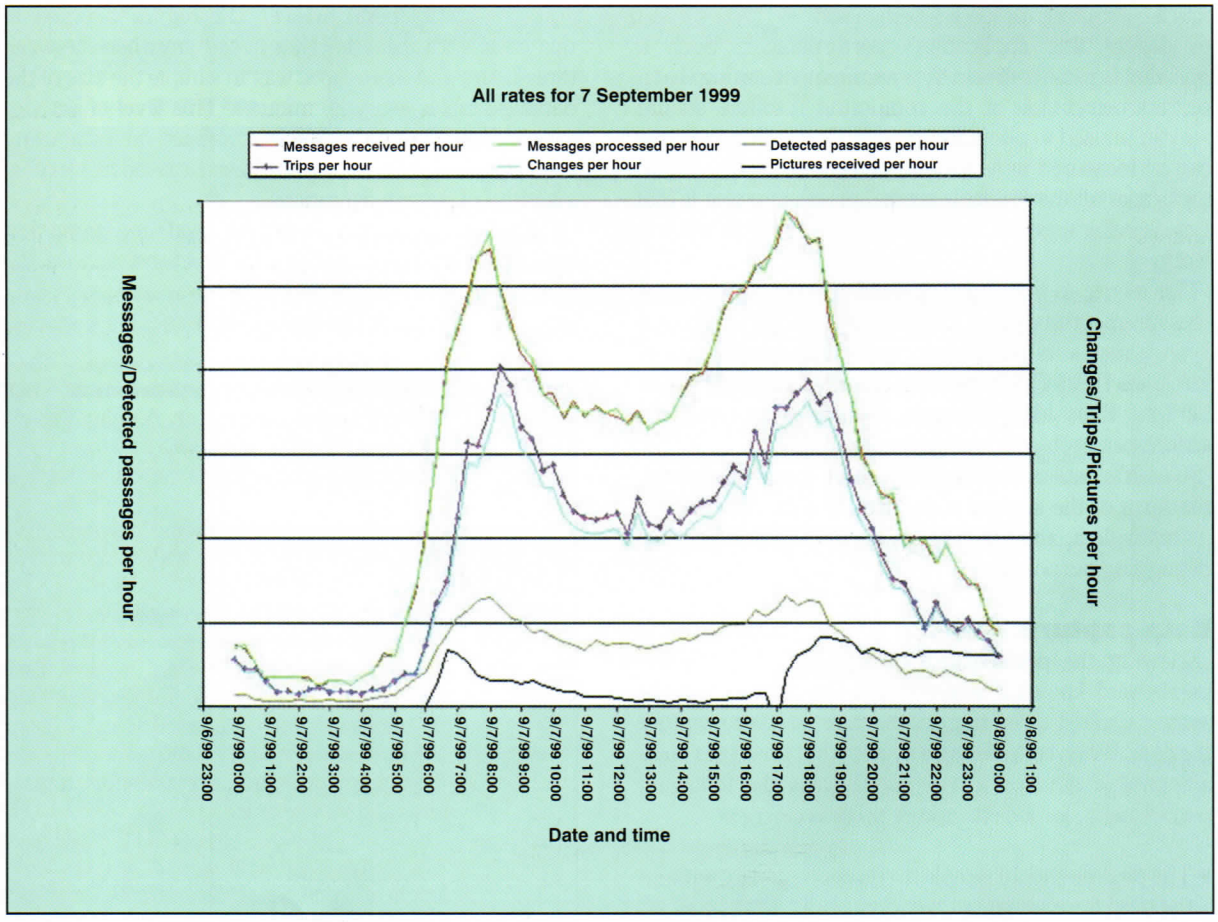

Fig. 6 Example chart showing a typical performance profile over a 24 hour period

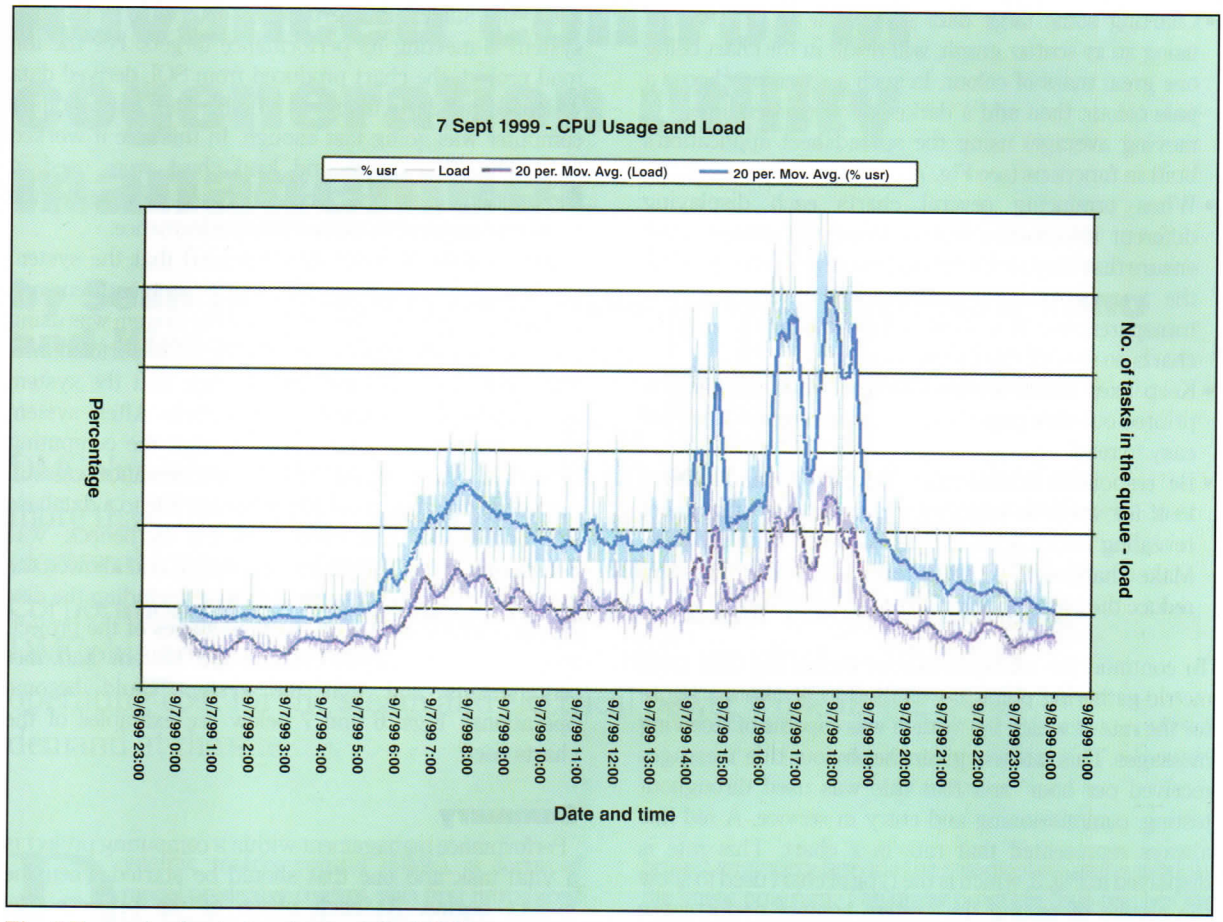

Fig. 7 Example chart showing a typical CPU usage and load profile over a 24 hour period

log files, custom database queries and other UNIX utilities.

Chart format

Results should always be accurate, unambiguous, and presented in a consistent way. It would be hard to argue against the accuracy of figures derived from UNIX utilities, but errors may be introduced during subsequent processing. In the SQL based data gathering process described above, it would be very easy to get hold of the wrong data, or get hold of data and then mis-label it. Calculating rates is another opportunity to introduce errors. So the rule is, if you use a process to produce result charts, test the process. Don't assume that the spreadsheet application, or a macro or some other function, works. Go through it and check.

Being unambiguous means being absolutely clear about what the data represents. A misleading data label is a very dangerous thing and can mean that extra work is done to solve a problem that doesn't exist, or a real problem is overlooked. In addition to getting data labels correct, it is advisable to mark on the chart the test conditions, version numbers of software and other relevant information. This enables users of the information to compare current results with the results of previous tests, on the basis of like versus like.

Consistency is paramount. When project progress is less than ideal, the same test may be repeated many times. The users of the charts will begin to form patterns in their own minds based on the colours and contours within the chart, i.e. they look for familiar shapes. They will seek to form opinions about the results without necessarily referring to scales on the axes or other relevant information. Consistency removes some of the potential for confusion and involves:

- Planning the chart—think about the purpose of the chart. What is the key message that is to be given? Make the chart a design activity in its own right. Look up the relative merits of pie charts versus bar charts versus *xy* scatter graphs, and then tie a format to a particular dataset.
- Choose a unique colour for each data item and stick with it.
- Name the data item accurately initially, then use that name throughout.
- For a series of charts covering a period of time that will be compared against each other, ensure that the scale of *y* axes is consistent, and that all axes are labelled consistently.

- Charting some large data sets, such as CPU usage, using an *xy* scatter graph, will result in the chart being one great mass of colour. In such a situation choose a pale colour, then add a darker coloured trendline (e.g. moving average) using the spreadsheet application's built in functions (see Fig. 7).
- When producing several charts each displaying different information but covering the same period, ensure that they share the same time period defined by the *x* axis. Imagine printing the charts out onto transparencies. Would it be possible to overlay the charts so that all the *x* axes lined up?
- Keep axes labels oriented so that when the chart is printed out onto paper and bound in a report, they are easy to read.
- Be responsive to feedback from users and always look for ways to improve presentation, so that more revealing information can be extracted from the data. Make changes to existing formats obvious, again to reduce the possibility for misunderstanding.

To continue the example started above, the data from metric gathering point ① was used to generate a figure for the rate at which the system was capable of receiving messages. This rate was given the obvious title 'messages received per hour' and that title was used throughout testing, commissioning and entry in service. A red line always represented that rate in a chart. This rate is displayed in Fig. 3, which is the type of chart used to show the results of testing the system against performance requirements such as SR-PERF-001 using a message generation simulator. Fig. 3 shows the requirement SR-PERF-001 represented as a broken horizontal line drawn from the *y* axis at the point representing the required rate. The message generation simulator is used to load the system at a rate above the requirement. The red line on the chart reveals whether the system meets the requirement, i.e. the red line should be above the requirement line. Comparing Fig. 3 with Fig. 6 reveals that the same type of chart is used both for simulation based testing and in service.

Automatic charting

A modern spreadsheet application such as MS Excel is a very powerful tool. Excel comes complete with a sophisticated Visual Basic editor and closely integrates the elements of sheets and charts with the programming language. The user interface to the automatic charting program developed for the toll road project is given in Fig. 4, and a summary of the macros in Fig. 5. The interface has a series of simple buttons to guide the user through the charting process. The process was split into discrete steps to allow the user to perform manual interventions or checks at any stage.

Using charts

The charts and other performance data are gathered to allow the stakeholder groups to determine whether a system is meeting its performance targets. For the toll road project, the chart produced from SQL derived data was used regularly to assess whether the central tolling computer was going fast enough. In this role it worked well. The CPU usage and load chart were used in conjunction with the first chart to determine the resources required to deliver this performance.

Initially a Fig. 6 type chart revealed that the system processing rates were not adequate to meet requirements, and a Fig. 7 type chart revealed that the system was using 80% of its CPU capacity to deliver that performance. This indicated, amongst other things, that the system had inadequate computing resources. After system modifications, new charts showed that the computing resources were adequate, but that performance was still not good enough to meet requirements due to a database I/O bottleneck. The source of the bottleneck was determined using the `iostat` utility, and allowed the developers to reorganise the database, including the disc striping arrangements. In the final stages of the project, they were used to show that the system had met requirements and that the system could become operational. Figs. 6 and 7 below are examples of the charts used.

Summary

Performance management within a computing project is a vital task and one that should be started when the project begins. At each phase of the project's life, consideration has to be given to:

- the performance requirements
- the system that delivers the performance
- the mechanisms used to measure that performance delivery
- the methods employed to visualise performance

so that people have the information they need to make the decisions regarding system acceptance. All four points must be addressed. To avoid any one is to create problems later which may be insurmountable.

The techniques described in this article did allow the project to meet its goal of getting into service a system that meets its performance requirements, and the tools developed are still in use for load monitoring purposes. If you are about to embark on a similar project, you cannot begin to address performance too soon.

Reference

1 COCKCROFT, A., and PETTIT, R.: 'Sun performance and tuning' (Sun Microsystems Press, 1998, 2nd Edition)

Rupert Brown is a Senior Consultant with Adacel Technologies Pty. Ltd., 250 Bay Street, Brighton, Victoria 3918, Australia, one of Australia's leading systems integration and software houses. E-mail: rupert@adacel.com.au. Web site: www.adacel.com