WOPR8 –Experience Report

Richard Leeke

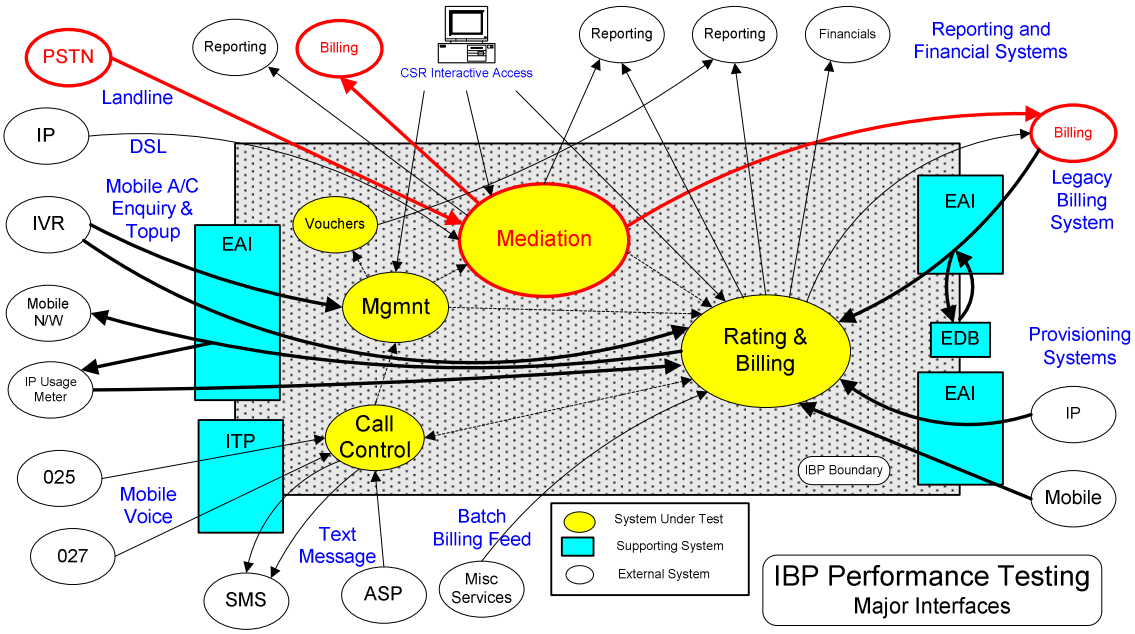
The Assignment

We were engaged to conduct a complex performance test for NZ's major telecommunications company. The customer was implementing a new integrated billing solution, to allow all services for a given customer to be combined for billing purposes. Initially this was to include billing for all PSTN, IP data, mobile voice and SMS services, with numerous other types of services to be supported over time.

The solution involved the integration of several systems from different suppliers, with high volume, real time interfaces between many of the systems. The performance test was required to confirm the behaviour of the integrated solution, and provide an ongoing performance regression testing capability. The scope of the testing included 9 distinct applications deployed over 12 platforms and involved emulating over 30 interfaces using 7 different protocols.

At the start of the engagement, the politics were "interesting", with key players expressing doubt over the viability of an integrated test such as this and proposing a much simplified approach (testing components in isolation, if at all).

The context diagram for the overall test is shown below, to give an impression of the complexity. The components and interfaces highlighted in red show the scope of the very first stage of the testing, which forms the subject of this case study. Clearly this was a small subset of the overall job, but it was not without its challenges.



Our First Deliverable

The first component of the solution was responsible for the "mediation" of telephone call data records (CDRs). (Mediation is essentially validation and reformatting of various formats of records before delivery to the billing system.)

The test suite simply had to generate valid and realistic data, deliver it to the system under test by FTP at an appropriate rate, and monitor the throughput and utilisation of the SUT.

This was probably the simplest component of the overall test, but the timescale was tight and there was some complexity involved in generating high volume, realistic test data in real time. The peak transaction rate was around 3.5 million call data records per hour (1,000 tps), and we needed to run headroom tests at perhaps twice that rate. The test data had to be generated in real time to include appropriate current timestamps, since there was significant time-dependent processing performed by the system under test.

As this was the first phase of a much larger, integrated test, we also decided to use this test as a proof of concept for many of the techniques we intended to use later. So we put quite a bit of work into the design and development of generic components, (at the risk of taking our eye off the ball with respect to the specific initial test deliverable).

The design goals included centralised control, monitoring and results collection, even though many aspects of the test could not be driven directly with the test tools we were using. It was also important that the time to restart a run (with the same or different control data) should be short. Our target was to be able to stop and start test runs within a maximum of a couple of minutes. We could not afford lengthy data preparation times between runs.

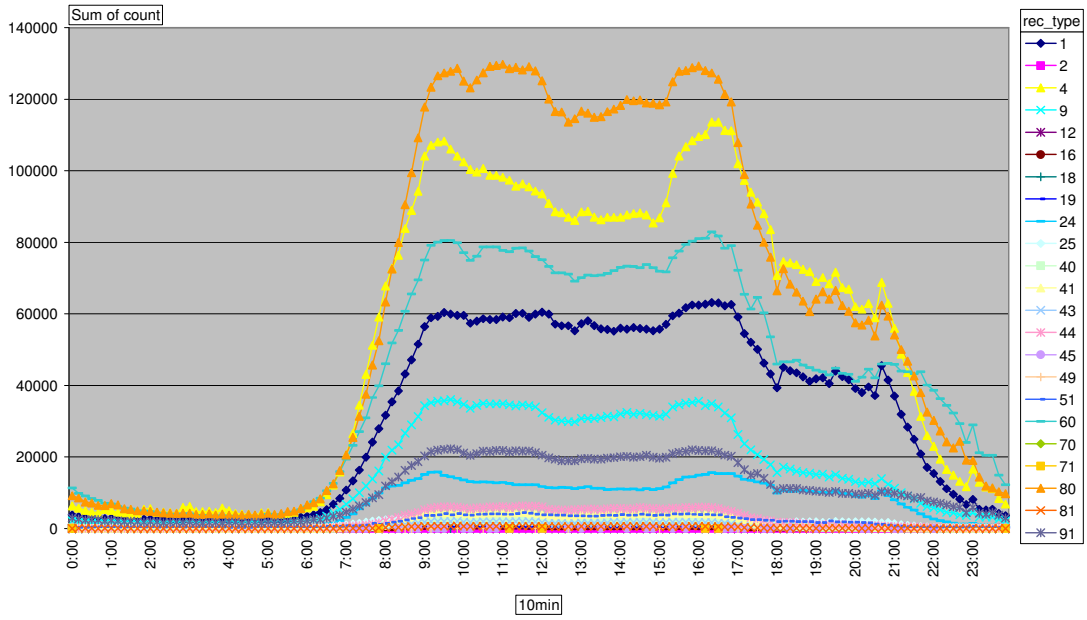
The approach we took to generating test data was to use a peak hour's worth of real production CDRs as source data, processing those in real time to deliver valid current data. This involved manipulating the call start and end date and time, as well as various fields such as per-exchange call sequence numbers (which were used as the basis for duplicate or missing record detection by the system under test).

The time manipulation was designed to allow the transaction rate to be scaled, to allow both headroom tests and realistic extended duration tests simulating the full business cycle (i.e. matching the call rate throughout the 24 hour period).

Although the system was initially being implemented with CDRs delivered by FTP (in 130,000 record files, every 2 minutes or so at peak times), the plan was to move to real-time CDR delivery, using a proprietary protocol, in a subsequent stage. The data manipulation process was therefore designed to work in either mode.

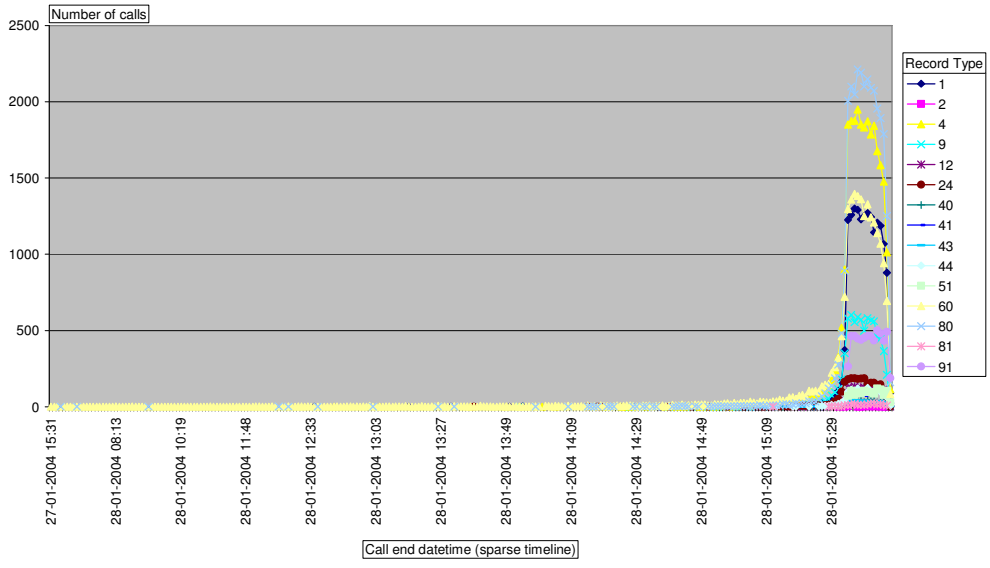
Due to the compressed timescale, most of the test development had to be done based solely on our interpretation of the specifications. We only got a sample of realistic data 3 days before test execution, and only got access to the system under test the night before the tests were to be run – so had no opportunity to test the tests.

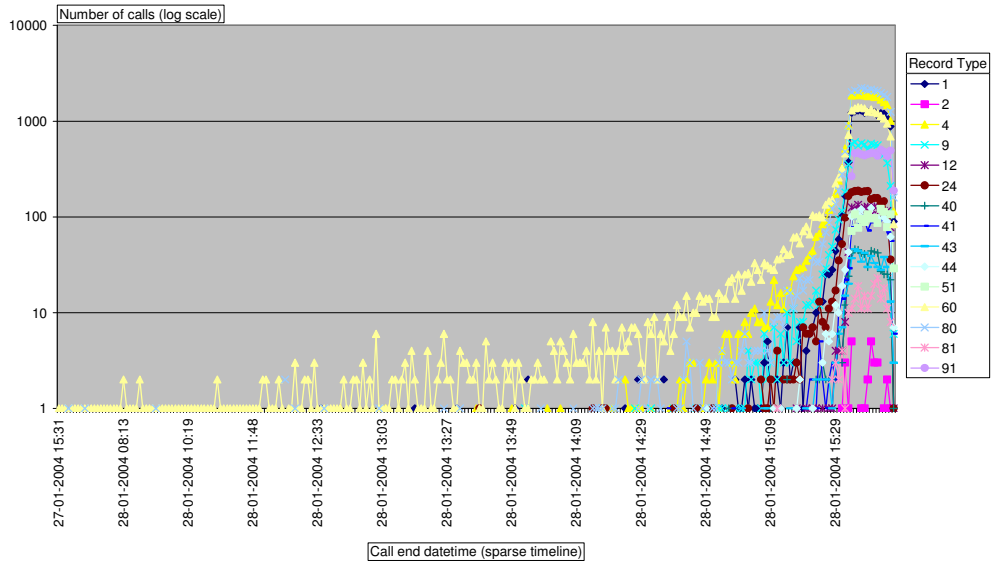
When we finally got the sample data, our initial analysis showed the sort of variations in load by time of day that we were expecting. There were some small variations in load profile according to the types of call, but generally the pattern looked straightforward. We chose an hour from the morning peak period to form the "seed" data for our data generation process.



Drilling down on an individual file, however, we found that there were a small number of "outlier" records which did not match assumptions we had made in manipulating the times in the test data. We had assumed (based on the specification) that all calls in a given CDR file would have ended within a few minutes before the file arrived for processing. This assumption simplified some of the timestamp manipulation slightly, but was to prove to be our undoing.

The charts below show the number of calls of each type, by call end time, for one sample file. A small number of call data records were delivered some hours after completion of the call (referred to as "late landing" usage records). The second chart, with a log scale, makes clear just what a small proportion of the data these were.





We were a little concerned that this skew in the distribution might be significant, so checked with both the project architect and the package vendor's on-site technical representative. Both assured us that such a tiny proportion of the records would not have a material impact on performance, so given the time pressure we opted not to make last minute changes to our data generation script to cater for the outliers.

We conducted the tests successfully, on schedule, with the system meeting all performance requirements. The only slightly intriguing anomaly we found at this stage was that the "partial failure" scenario, in which one of the two nodes in the system under test cluster was disabled, produced substantially higher throughput than the normal mode test. The clustered solution was intended to provide load balancing and redundancy – but in practice a warm-standby configuration would have been better.

What Happened when the Solution went Live

About three days after the system went live, there was a total meltdown. The system started consuming 100% CPU and no CDR files were successfully processed. The upstream system delivering the billing records were sized to cope with an outage of around two days – after which billing records would start to be lost (and hence the customer's revenue stream would be turned off).

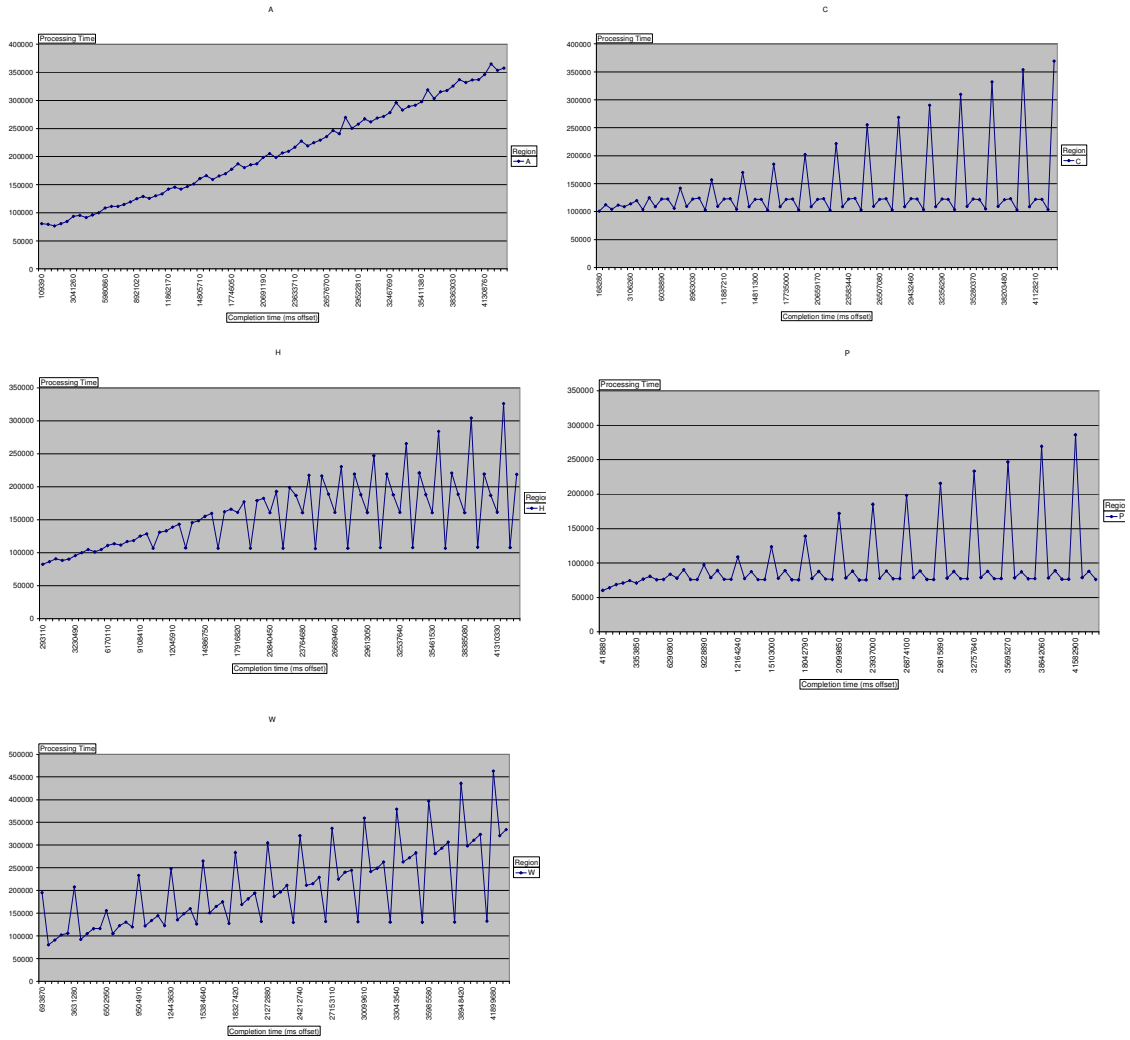
A 24 hour crisis team was established, vendor experts were flown in from overseas, racing against time to find what was going on before billing data was lost. A temporary work-around was identified, by relaxing some of the business rules, but the root cause of the issue proved elusive.

Naturally our credibility took a huge hit, with the sceptics gleefully pointing out that if we couldn't even deliver useful results for something as simple as this, what hope was there for the complex parts of the solution.

We eventually managed to salvage some credibility by correctly speculating that the root cause of the problem was those outlier records we had ignored in our tests. This suggestion was initially dismissed by the vendor, but after enhancing our test suite to emulate the data

distribution correctly we were able to demonstrate the cause of the problem in the test environment.

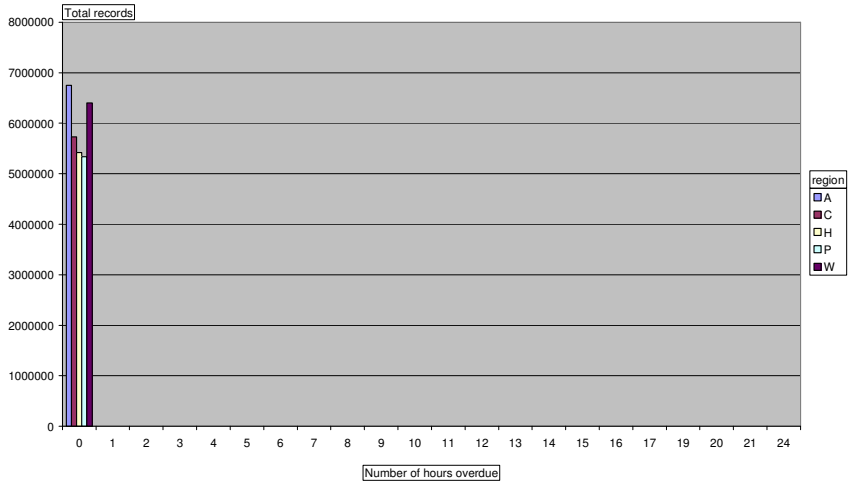
We took a sample of an hour's worth of files – which was 5 files from each of 5 regions, and processed generated files based on these 25 source files continuously for an extended period, simulating a day's worth of processing. The charts below show the response times for each region. Each dot represents the time to process one file.

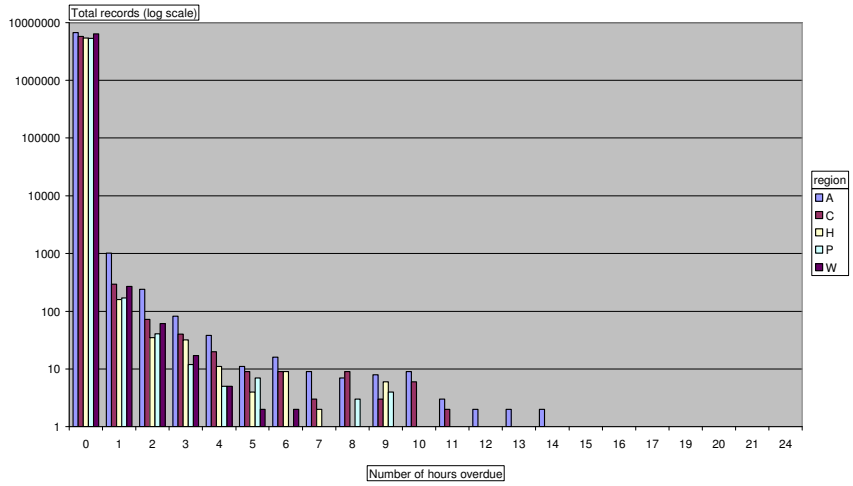


As the test run continued, the elapsed time to process some of the files grew longer and longer, whilst the time for other files remained constant. Strangely, the different regions showed very different performance patterns, with files derived from all 5 source files for one region getting slower at about the same rate, whilst the other regions had some files which slowed down and others which didn't. Different source files also resulted in different rates of slow-down.

We then analysed the time distribution of records in each of the 25 files, and showed that the files which slowed down were the ones with "late landing" usage records. The more overdue the earliest record in the file, the more the file was delayed. We also established that the increasing response times were due to the increasing volume of history to be checked against as the test progressed.

Applying the same analysis to the production data causing the problem, we established that if a file contained a record that was more than about 24 hours overdue, this resulted in the system becoming completely blocked. The charts below (normal and log scales) show the number of overdue records in 24 hours worth of data (around 30,000,000 records). In this case there was one record that was 24 hours overdue, and a total of thirteen that were 12 hours or more overdue. This day probably wouldn't have been bad enough to trigger the meltdown.





Explanation of the Behaviour

It turned out that the system's duplicate checking worked as follows.

1. CDR files were processed in parallel, with "n" streams configured. Files were dispatched to the next available processing stream.
2. For each CDR file, scan the file to establish the earliest and latest call end date and time.
3. For each record in the file (typically 130,000 records), check for a duplicate record in every prior CDR file with an overlapping time range. So for a file with a 24 hour overdue record, the duplicate check would be performed against the previous 300 files, or 34 million records.
4. If processing of a given file did not complete within a timeout period (configured to be 20 minutes), the system assumed that the job had failed and resubmitted the file to

the next available stream. When a timed out process finally finished its results were ignored.

So once a file was received that required longer than the timeout period to process, more and more streams ended up trying to process a copy of the file, only to have their efforts discarded. Eventually both hosts in the cluster were running at 100% CPU utilisation, with all processing streams stuck on the same file.

Resolution

Several changes were identified to solve the issue.

1. As a temporary measure, the maximum time window to check for duplicates was reduced from three days to three hours. This kept the system going while the root cause was investigated.
2. Subsequently, the system was reconfigured such that files for each of the five regions were processed independently, instead of all being treated as a single pool. There was no point in checking for duplicates against other region's files, since the aggregation of CDRs in the network meant that all data from any given exchange always ended up in the same region's files. This reduced the volume of records to check for duplicates by 80%.
3. The package supported two duplicate checking algorithms – the file based approach that had been in use gave the highest throughput as long as the data was very tightly distributed in time. An alternative, record based approach was recommended for less tightly clustered data such as ours.

The actual root cause of the late landing usage was never definitively identified, although I have a theory. The issue only seemed to happen with certain telephone exchanges. All of the late landing records came from a small number of old exchanges, of similar models. My hunch was that during a period of declining call volumes, a port on the exchange could get "stuck" on completion of a call. The CDR for that call would not be delivered until traffic increased enough that another call used that port. Discussing that with the telephony experts this seemed plausible – however given the age of the equipment and the fact that the problem was resolved, this wasn't pursued further.

And the Moral of the Story is...

We definitely re-learned a few of the golden rules of performance testing from this experience:

- Avoid simplifying assumptions whenever possible – and if you have to simplify, make sure you understand the architectural implications
- Don't take the system experts' words for it on questions of architectural significance
- Pay attention to the politics