Getting the System Sizing and Performance Test Right

Steve Hazeltine - Sema Group Systems Ltd. (now Oracle UK Ltd.)
(Email shazelti@uk.oracle.co - Telephone +44 171 830 4444)

Andrew Johnston - Independent Consultant and Sema Group Associate
(Email andrewj@compuserve.com - Telephone +44 1483 283408)

1. Introduction

This document describes how we specified the Sun server required for the ported Rental Systems at Livingston UK. Livingston is the European leader in providing services to users of electronic equipment and computers. A primary business activity is the rental of electronic equipment and computers, much of which is conducted over the telephone, so response times are key to the business. We therefore had to be sure that the new system would perform at least as well as the old one without wasting money through over-specified hardware.

The exercise was a complete success. We were able to reduce our original hardware estimates by over £200,000 from the manufacturer's recommendations, and from day one the system was faster than its replacement. The observed CPU and memory usage figures demonstrate that we procured exactly the right specification for the client. Our approach had a number of additional benefits to Livingston, including being able to accurately predict, and thus reduce, the number of Oracle licenses required.

The key to the success was:

- understanding the importance of performance to the client, and constructing an approach which met their needs,
- taking measurements from the old system to generate the real user requirements and transaction volumes,
- experienced staff using this data to select the required platform,
- using a combination of a simulator and live users to provide a realistic performance test, to validate our approach before proceeding to live running.

Because it relies on having an existing system in place, this approach is not applicable to every situation. Today, however, the replacement of an old system is probably more common than the creation of a completely fresh application, so we hope our experiences will be of some use to others.

2. Approach to Specifying the Hardware

There are a number of approaches to hardware specification, including:

- approximate sizing based on the number of users, and the recommendations of hardware manufacturers,
- transaction processing and other benchmark based sizing,

- comparison with similar existing systems,
- prototype based benchmarking.

The choice of approach should be based on:

- understanding the various options,
- the cost of estimating the size of the platform,
- the risks to the supplier and client of mis-sizing the platform.

Cost is important. A prototyping exercise is not cheap, and for a small number of users it may be better to make an approximate sizing calculation and over-specify the hardware. Users rarely complain about a system running too quickly.

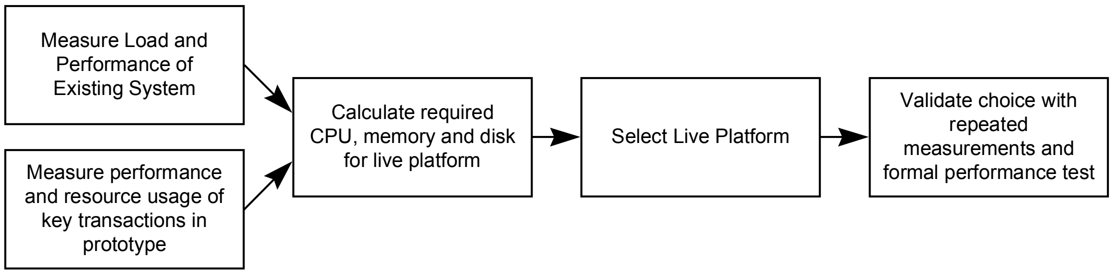
For a larger system, the risks need to be understood, and their cost evaluated. If an external supplier is involved, both parties need to be clear on who will bear the cost if the risks materialise. These could include:

- the cost of a hardware upgrade, particularly significant if the selected model is at the top of a range,
- cutting back to the original system, and repeating the system implementation,
- consequential loss of business and client's image,
- loss of faith in the project by users, who may not be willing to give the IT department or supplier a second chance.

We had the following situation:

- performance was critical to sales users,
- cutting back to the old system would have caused immense disruption to the business, with potential loss of sales,
- previous experiences had been very painful, both for individuals and the business,
- we had conflicting signals from our initial assessment of the transaction rate (which suggested quite a small system), and the advice of the hardware manufacturers based on the number of users, which suggested a very much larger system.

We had to get it right first time, and it was clear we could not rely on previous experience and manufacturers' data alone, so a prototype based approach was selected. The process we adopted is summarised in the figure below:


Measure Load and Performance of Existing System
Measure performance and resource usage of key transactions in prototype
Calculate required CPU, memory and disk for live platform
Select Live Platform
Validate choice with repeated measurements and formal performance test

3. Measuring the Transaction Load

3.1 Identifying the Key Transactions

We started the sizing exercise with very little reliable data:

- The number of concurrent users was high (potentially > 100), which suggested a large system,
- The number of enquiries and orders processed per day was relatively low, suggesting a much smaller system.

We therefore realised we had to find some key indicators which would allow us to specify the performance much more accurately, with the following characteristics:

- The old and new system architectures were entirely different (moving from a proprietary database on Data General hardware running AOS/VS, to an Oracle7 database running under Unix). Therefore low-level figures related to physical system attributes (e.g. %CPU loading) could not easily be compared between the architectures.
- Very high-level business transactions (such as the number of new orders per day) would be useful in comparing total volumes of business, but not much use for measuring subjective performance which is related to individual processing actions.
- The key transaction timings would have to be meaningful (to allow them to be related to the users' requirements), and accurately and easily measurable (for an exactly comparable process) on both the existing and ported system.
- We would not have a complete ported system until quite late in the project, so we would have to size the system using a relatively small subset of the transactions.
- The transactions would have to be typical of live usage. In most systems, a small group of functions are used most of the time, and these should form the basis of the prototype. Furthermore, you need to simulate a representative mix of read and write accesses, across a representative section of the database and disk.

Fortunately, we identified a small group of transactions (a cycle through from a sales enquiry to order confirmation) which were supported by the early pilot of the porting exercise and which satisfied these criteria very well. The cycle contained key transactions for which the users had specified target response times: product queries, customer queries, and the time taken to finish processing a confirmed order. The same enquiry screens could generate query-only transactions, allowing the mix of query and update accesses to the database to be controlled.

3.2 Measuring the Existing Usage

We started by verifying the number of active users on the system. An initial manual assessment (walking around at a busy time with a clipboard) suggested that the number of users actively loading the system was a very small percentage of those logged in:

Activity	**No.**
In Screen & Active	6
Screen (Passive)	11
Screen (Inactive)	15
At Menu	20
Operators/testers	9
Total	**61**

We tried to distinguish between those users obviously working from the data on the screen (passive) and those where the system had been left sitting in a screen but there was not obviously anyone using the data, or there was no data on screen (inactive). Allowing for errors, the number of active (and passive) screen users was around 17 ± 4 (out of 60+ logged on). This could not only account for the conflict between the initial estimates, but would also suggest much lower licensing and memory requirements. However, the figures were perhaps too low to be believed, and needed to be confirmed more scientifically.

We also needed to measure the numbers of each key transaction and the actual performance delivered by the old system. A short attempt at using manual measurements proved only that the transaction timings varied considerably, and the mix of transactions in the system was much broader than we had been lead to believe, so we could not just generate enquiries and order confirmations and hope this would mimic live use.

Fortunately the Data General system provided a rather elegant solution to our problems, in the form of an optional log of all screen activity for each user session during a time period. We could then replay these logs and observe both what the users had been doing, and the times taken for key transactions. Similar mechanisms exist in most operating systems, or can be simulated using standard test tools.

The only remaining problem was the volume of data this would generate. In the end we used our test tool (see the next chapter) to automate replaying the logs and to assess which function, on which screen, each user was performing. All such combinations were classified as a database read, database update, or “ignored” action (such as navigation from screen back to menu). As we replayed the logs the time stamps were used to both time key transactions and assess when user sessions were left inactive for a period of time.

The figures from these measurements broadly supported the earlier estimates, but with a few interesting aspects:

- There was a wide variation in the areas and functions used, with no obvious “80-20” rule,
- The number of “active” users was slightly higher than the manual measurements had suggested, but still less than half those logged on at any time,
- Database query actions outstripped those which would generate an update by a factor of around 2.5:1,
- Most key transactions on the old system were completed within the target times, but with quite a wide variation.

3.3 Establishing the Real Targets

Having established an accurate analysis of the number of transactions of various types throughout the day (on a few chosen days), we then had to set the targets for the new system. There were a number of adjustment factors which each had to be taken into account:

- Growth - we had to size the system to allow for a few years in a growing business,
- Other uses - we had to consider the increased system loading which might come from the introduction of more flexible reporting tools, better interfaces with the OA software, possible access from other European sites, network management and system monitoring and a number of other such changes,
- Operator and developer activity - command line sessions were not captured by the logging tool,
- Business volume - the measurements were taken during a “quiet” month in business terms (a normal seasonal variation due to summer holidays). We couldn’t wait for business to pick up, but we could compare the number of orders being processed on the measurement days with the peak values from a few months earlier,
- Greater spread of functions - we could only simulate a subset of transactions, and a greater spread in reality would undoubtedly decrease performance due to, for example, reduced data cacheing,
- Losses in the measurement process - for example, we had to break the logging down into one hour sessions, and lost about seven minutes resetting the system before continuing with the next. This inevitably disturbed the analysis of things like inactive sessions and the daily totals of transactions,

The worst-case number of active users was calculated as follows:

Peak from Automated Measurements	21
Add maximum error on measurements	4
Add 60% for business volume adjustment	15
Add 2 for operator/developer access	2
Maximum Concurrent Users	**42**

We also decided that the new server would have to support a load equivalent to roughly 3700 query transactions per hour, derived as follows:

Maximum load observed in tests	1276
Allow 10% for measurement errors	128
Add 20% for operator/developer access	255
Subtotal	1659
Business adjustment (ratio of measured business to past peak) of 50% gives	2488
Allow growth factor of 50% to give	3732
Target query transaction rate	**3732**

From the observed split between query and update transactions, we noted that the test load would have to include at least 1500 update transactions per hour. The requirement was that at this level, 90% of product and customer queries and 90% of order confirmations should complete within the defined targets. Our measurements also confirmed that these would be a small improvement compared with the existing system.

4. Building and Measuring the Prototype

In order to run a prototyping/benchmarking exercise of this sort, you have to do several things:

1. Choose, source and set up the hardware,
2. Choose and develop / set up the test tools,
3. Develop or port key parts of the target system,
4. Measure the performance of the prototype,
5. Specify the real platform,
6. Plan and run the performance tests.

You may be constrained by decisions which have already been made. In our case, these constraints existed but actually made the job slightly easier. It had already been decided that the target system would be a Sun server, running Solaris and Oracle7, because of Sema Group's previous good experience of this environment and Livingston's business relationship with Sun. Furthermore, the porting exercise was already under way, so we had part of the system, albeit incomplete and partially tested, to work with from a very early stage.

4.1 Choosing the Prototype Environment

You have to have access to adequate hardware and software to make the exercise meaningful. There are various obvious sources: development equipment, sale or return agreements, and loans from the manufacturers may be feasible, but purchase is probably the last option. Given the nature of our client, we went for the other obvious solution - rental. This is a very good solution and allows the exploration of a number of options, within limits.

Unless your test environment is exactly what you plan for live use, you then have to decide what is reasonable to extrapolate from these tests, and what should not be extrapolated. In our case, we knew that we would be using the same CPU architecture and we were confident that we could use the manufacturer's CPU performance figures to compare one model with another. On the other hand, we had decided for resilience reasons that the live system would use a RAID disk array, but none was available for us to test. Since we were confident the system would not be disk bound we decided to risk extrapolating figures from a different disk architecture, but this might not have been wise in a disk-bound system.

You also need to provide a sufficient platform to run the test tool(s) themselves. As well as a Sun SparcStation running the target application, we needed 4 Pentium PCs running Windows 95 dedicated to controlling the tests, simulating the load and storing the results. As a general rule you should try to avoid test tools which run on the target machine itself unless you are confident you can assess their own impact on performance.

4.2 Choosing / Developing the Test Tools

The choice of test tools varies with the application. We identified a number of key requirements related to our situation:

- The application was presented through a character-mode terminal emulator under Windows, so our test tool had to work with that environment,
- To allow variations in the tests depending on the response of the target system, the test tool had to include a scripting language of some sort,
- The tool had to be able to interpret the screen of the terminal emulator as an 80*25 array, not just a text control with a string in it,
- Given the large volume of data, the tool had to be able to write results away to structured

storage, ideally a PC database,

- We would have to be able to run several test sessions simultaneously from a single PC, to make running a sufficient number of simulated sessions practicable.

Surprisingly, we could not find a commercial test tool which met these requirements. The main problem seemed to be the ability to interpret text within a terminal emulator window, but the ability to write results to a database was also lacking in many options we looked at.

Thus we decided to write our own! While this should not be undertaken lightly, in our case it turned out to be a very effective solution. We used Visual Basic, which could use DDE to communicate intelligently with the emulator, could write its results directly to an Access database, and used Windows messages to synchronise between different test processes. The initial scripts we wrote to interpret the logs from the Data General provided most of the core functions to decide what the application was doing, and it was then quite simple to join these together with new code driving the application itself.

We also developed a related tool to capture keystrokes and commentary text, so that experts on the application could develop the framework of each test without needing any expertise with the test tool itself. We found that a test cycle covering about 10 screens could be recorded and developed into a working test routine in about 1-2 days.

Since (with the level of automation) it was easy to do so, we repeated the performance tests on several occasions: when the chosen live platform became available, with various specification client PCs, and when the ported system was functionally complete. This built up confidence in our decisions, and pointed the way to some performance improvements in the application, and the correct client PC specification.

4.3 Measuring the Performance of the Prototype

The performance load simulation was a routine which drove the ported application in a repeating cycle of enquiry, order confirmation, some general enquiries, deleting the previously created order, and some more general enquiries. The cycle, various product and customer searches, and the order confirmation process were each timed and results written to the results database. Analysing this routine in the same way as the existing system showed that each cycle generated 54 query transactions and 23 updates.

The test PC could drive up to eight “threads” - sessions on the target system. There were four parameters which affected the actual load generated:

- Number of test threads,
- The “Short Delay” - a delay of a few seconds inserted between each group of a few keystrokes to simulate natural operator entry speed (and allow the system to respond),
- The “Long Delay” - a maximum delay (in seconds) between repetitions of the cycle,
- Number of repetitions - this mainly affected the duration of a test to make sure that the results were statistically viable.

By varying these factors we could simulate widely varying test loads. The behaviour of the prototype system was then assessed using three factors:

- The number of query transactions per hour (derived from the average order cycle time and the number of threads running),

- Whether the timed transactions completed in a satisfactory time,
- Whether the target Unix host's performance monitoring tools reported saturation of CPU and/or disk I/O.

The following table summarises the results of our prototype test against our development SparcStation (a 110MHz Sparc 5):

Test Run	**Number of Threads**	**Short Delay**	**Long Delay**	**Cycle Time**	**Query Tx per Hour**	**Product Search**	**Customer Search**	**Order Confirm**	**CPU Usage**
927	1	1	60	00:02:41	1207	1.3	0.3	3.6	20-35%
933	2	1	60	00:03:42	1751	1.5	0.3	5.0	40-70%
928	2	3	60	00:05:43	1134	1.7	0.2	3.6	30-60%
932	2	5	60	00:07:59	812	1.5	0.2	4.3	10-50%
931	3	3	60	00:06:17	1547	1.8	0.3	5.4	40-70%
930	4	3	60	00:07:10	1808	2.3	0.4	4.9	50-80%

The disk I/O rarely exceeded 15% of maximum throughout all the tests, but the 3 and 4 thread figures showed CPU usage approaching saturation, and the 4 thread figures showed search times starting to exceed the target. From this we concluded the following:

- performance should not be disk bound, but we were probably getting unrealistic data cacheing and would need a better spread of functions in the final performance test,
- The SparcStation 5 could support up to about 1750 transactions per hour (keeping machine usage and transaction times within targets), so the CPU of the live machine would have to be about 2.5 times this powerful.

5. Specifying the Live Platform

The aim is to produce a platform specification in terms of model number, number and speed of CPUs, memory size, disk architecture and capacity. The inputs to this process are:

- the CPU, memory and disk resources consumed by the prototype,
- the transaction load and response times supported by the prototype,
- the required response time and transaction load,
- resilience requirements (which may dictate extra CPU or memory capacity, or a certain disk architecture),
- memory and disk sizing guidelines from hardware and / or database suppliers.

The following sections provide some guidelines, especially for processor sizing, based on our experiences.

5.1 Identifying the Bottlenecks

You must review your prototype to ensure that there are no exceptional circumstances, such as the database being very poorly tuned, or the test hardware being so short on memory that swapping and paging occurred at very low loads. If the test hardware can comfortably support the expected live load, you may be able to reduce the specification. Otherwise, you will have to identify the bottlenecks in performance, and hence decide what needs to change:

- With a lightly loaded machine running only a few transaction types (where most data will be cached) the response time represents the time for one processor and the disks to complete one transaction. You should see little disk read activity, and CPU activity roughly proportional to the transaction load. Increasing the number of processors will **not** improve the individual response times, but increasing individual processor power/ speed will. In practice, available processor speeds typically vary by less than a factor of 2:1 at any one time. Therefore, if the response time of a lightly loaded prototype is inadequate, the cure is to modify the software design, not to expect great improvements from faster hardware.
- If as the load increases, the disk activity remains acceptable, response times will increase as the probability of any one process having to wait for other active processes increases. If the processor load gets too high (generally above 75-80% on most Unix systems) excessive swapping will cause the response to worsen dramatically. The onset of this can be delayed by faster or additional processors.
- If disk performance is the bottleneck, this will show itself either as an inability to fully load the CPUs when running a higher transaction load, or a dramatically worsening response as the mix of transactions is increased to reduce cacheing. You may need to look at alternative disk architectures, very much more memory (to extend the effects of data cacheing), or software optimisation.
- If under the same circumstances paging activity becomes very high then the system is memory bound. You will have to specify a larger amount of memory for the real system, and extrapolate from more lightly loaded figures. Alternatively, memory is one resource which you should be able to rent or borrow to check the effects of a larger quantity. It is usually a false economy to under-specify the memory on a machine where the CPU and disk can support better performance.

In our case, we found that disk and memory activity were acceptable up to the limit of the transaction load our test machine's CPU could support. We were therefore able to specify the live machine as a proportional increase in CPU and memory capacity, with a disk architecture derived from the resilience requirements.

5.2 Selecting the Processor(s)

The following shows how to:

- select a processor speed that ensures that a lightly loaded machine will give an adequate response time,
- calculate how many processors you need to support the expected peak transaction load.

We use the following terms:

c_p	CPU rating (e.g. clock speed) of the prototype's processor
c_r	CPU rating (e.g. clock speed) of the live processor

f_c	Adjustment for the effects of a real load compared with the higher cacheing of a typical simulated load
P	Number of processors in the server
r_p	measured prototype response time of the most critical transaction (e.g. with the slowest execution speed)
r_r	required response time of the same transaction in 90% of cases
t_p	transaction load supported (at <80% CPU Loading) by the prototype
t_r	peak transaction load predicted for the live system, including any growth or adjustment factors

You can make an initial estimate of the required processor power/speed as:

$$c_r >= \frac{1.1 c_p r_p}{r_r}$$

The factor of 1.1 allows for an expected increase in average response time when the processor has a reasonable (40%) loading. If a processor of this speed/power is not available, the remedy lies in changing the software, not in hardware.

The required number of processors is then given by the ratio of the peak transaction rate to the transaction rate supported (at a reasonable CPU loading) by the prototype:

$$P >= \frac{f_c c_p t_r}{c_r t_p}$$

This gives the (fractional) number of CPUs that would be fully loaded by the system. f_c is a factor to allow for non-key transactions not included in the prototype, and other aspects which will make the real transaction mix more heterogeneous than simulated in the prototype.

In our case, we found that a small number of transactions accounted for around 60% of the load, and treated the remainder as being equivalent in total to the main sales enquiry transaction. Thus we estimated a value of 1.5 for f_c.

The next step is to compare potential processor configurations from the manufacturer's literature that will meet the performance goal. For each potential configuration, you should use the above equations to check that the response time will be acceptable and the supported peak transaction rate adequate:

$$\frac{1.1 c_p r_p}{c_r} <= r_r \quad \text{and} \quad \frac{c_r t_p P}{f_c c_p} >= t_r$$

Given a range of acceptable alternatives, your final choice will then be based on:

- price,
- relative expected response times,
- likely transaction load growth over the project lifetime.

Our Project Experience

In our project, we had already decided on Sun Microsystems as the hardware supplier.

The measurements on the existing system showed two peak loading times: around 10-12 noon when sales activity was at a peak, and early evening when despatch was processing the day's orders (but sales activity was reduced). We calculated the load for both cases, and concluded that the greatest CPU usage was in the morning. The key transaction for response time had been identified as a product enquiry.

The prototype measurements and above calculations showed that:

- the minimum individual CPU power could not be less than 75% that of a 110Mhz Sparc 5,
- the total CPU power should be at least 2.5 times that of the 110Mhz Sparc 5.

This immediately allowed us to consider the less expensive Sparc 20 range rather than the more expensive Sparc 1000 (capable of taking significantly more processors). Without the prototype, we would not have been able to make this economy with confidence. There were three main options:

1. two 50MHz processors
2. four 50MHz processors
3. two 70MHz processors

Of these, the loading for the 2x50MHz configuration was too high for comfort. A 4x50MHz unit would have been able to support increased throughput, and therefore support greater short-term organisational growth, but the 2x70MHz option offered better response times, with better expansion potential in the medium to long term. The decision was therefore taken to procure that configuration, as a fast response time in the short to medium term was more important than short-term growth. This would support the business better, even though it had a lower specification in simple Specmarks.

5.3 Memory Sizing

The guidelines for memory sizing are usually related to the number of users. As we noted above, many of the users on the old Livingston system were logged in for long periods but performed relatively few transactions. We could therefore use as a basis the measured number of active, rather than concurrent users. Armed with this number we started with the standard guidelines for a typical Unix/Oracle installation:

Source	**Memory (Mb)**
Unix	32
Live database: 40 active users @ 3Mb each	120
Test database	16
CQCS 4GL: 40 users @ 0.5Mb Each	20
TOTAL	**188**

From experience, we were aware that memory size is critical in Oracle performance, and any extra memory would be useful. We also had to allow for growth and other processes (such as monitoring software). We therefore decided to buy 256Mb of RAM, safe in the knowledge that if this was substantially over-specified any extra could be deployed on another installation.

Where a decision like this is borderline, but good performance is essential from the outset, the rental option is very practical. Extra memory can be rented, and the system should go live with this additional memory in place. Use the performance measuring tools to decide if the memory is being used. If not, return the excess to the rental company. If it is needed, extend the rental until new memory can be bought. This guarantees performance at a much lower cost than outright purchase.

5.4 Specifying the Disks

Sizing the Disks

Disk sizing is covered in standard texts and manuals (e.g. the DBA guide for Oracle), and will not be covered here, except for a couple of salutary tales:

Our first estimate of disk sizing was based on two good numbers: an estimate based on the known numbers of rows in each table in the old system, with a factor of three for indexing; and some knowledge of the disk sizes in the old Data General system. We were therefore rather surprised to find that the first-cut import into Oracle occupied over three times this volume of disk, or almost ten times the size of the raw data. This was due to a number of factors:

- A very denormalised data structure,
- Larger archive tables than we had been led to believe,
- A data structure enforced by the 4GL in which most fields in the database were of fixed length and “NOT NULL”, even if usually empty. Most of our new database was empty space!

This type of problem is rather insidious: it can happen by default as the result of a low-level decision outside your direct control, and it will have a major cumulative effect. In our case, if affects not only database sizing, but also performance, size and speed of backups, and so on. To limit any risk to the porting exercise, we decided to live with the structure in the short-term, but are now progressively making manual changes to reduce this effect.

Other factors to consider in disk sizing are the following:

- User files and application software *will* grow in size, no matter how good your intentions to control them. This is particularly true in the PC arena.
- You will lose more to formatting and the operating system than you predict.

  In our case, we had decided on a RAID array for the main disks of the live system, and found that we lost 10-20% to raw disk formatting, and another 20% to the RAID operations, so that the available size is only about 65% of the total. Other operating systems have different “features”, but with similar effects.

The message is simple: be generous in sizing your disks, or make sure you have an architecture in which you can easily at least double your disk capacity if required.

Estimating the Number and Speed of Disks

To ensure performance, it is important to have enough disks of sufficient speed available. While running the prototype, you should run a disk monitoring utility (*sar* or *perfmeter* are adequate under Unix). You should ensure that your prototype database is sufficiently large and choice of data sufficiently random to make the data cacheing as realistic as possible.

You will need the following data from the prototype:

- disk usage (typically a percentage) during a fixed period,
- number of physical and logical reads and writes over the same period.

Look at the ratio of logical to physical reads and writes. It is reasonable for small tables and indexes to be cached, and you are trying to tune for the best ratio possible. However, if you get a ratio in excess of 4:1 you are probably getting more cacheing than you will see in real use.

From the manufacturers statistics you should be able to calculate the approximate time taken to read or write a block of data, D:

$$D = (\text{mean seek time}) + 0.5*(\text{rotation time}) + (\text{block size})/(\text{peak transfer rate})$$

As a confidence check, you may be able to see a limit to the number of **physical** reads or writes per second when the system is heavily loaded: this will be roughly 1/D.

Given the processing times D_p and D_r for the prototype and proposed live disk systems respectively, the steps are very similar to those for processor specification.

1. Review the degree of caching in the prototype compared to the live system. To do this, consider which tables and indexes should be cached, and refer to the code in your prototype. Consider also how much spare memory you may have.
2. Ensure that the live system's disk processing time satisfies $(D_r/D_p) < (r_r/r_p)$, otherwise you may have a problem with individual transaction times even if the overall transaction load can be handled.
3. Calculate the number of disks *N* required to handle the peak transaction load in a fashion analogous to the calculation of number of processors:

$$N >= \frac{f_c D_r t_r}{D_p t_p}$$

Note that because disk are mechanical devices, they are slow, and that additional disk delays can have a very marked effect on performance. Most manufacturers recommend that disk loading in operational systems is kept under 40%. To achieve this, we would recommend at this stage aiming for a loading of under 25% - so take a value of t_p where this is true.

4. In your database design and when setting up the system, stripe or distribute the live database across at least *N* spindles, so that each is equally used, and so that none of those disks form a bottleneck.
5. Remember to consider the reliability aspects. Note the manufacturer's guidelines and your standards for location of operating system software. In particular, you may want to ensure that transaction logging uses a separate disk to ensure that the database can be recovered in case of a disk crash. You may need to allow for separate disks for this, and you may want to stripe it across several disks to ensure adequate throughput to the logs. However, in our experience with Oracle one logging disk is probably sufficient unless you have a very heavy transaction processing system.

In our project, we found that disk loading was usually very low. For reliability purposes we went for a RAID array in which the disk seek times were adequate, and throughput from the RAID array was much higher than we required due to the striping from the RAID 5 configuration employed. Note that because our system is dominated by query transactions, the reduction in performance from a RAID 5 configuration is not a problem. In cases where a higher proportion of transactions are updates, other disk configurations might be more appropriate.

6. Doing the Real Performance Test

Before live use, we carried out a performance test which would be more representative of live operation. This was based on the automated test, but with the following changes:

- the original test cycle was supplemented by two more to exercise other parts of the system and reduce the effects of data cacheing,
- a large number of sessions were simulated in which very few transactions were generated, but the system was navigated from screen to screen at random to simulate the expected "inactive" and "passive" users,
- a few users were invited performed a few transactions while the system was under load, to verify that the observed performance was in fact acceptable.

The automated test suite was used to generate a load roughly equivalent to the predicted peak (including an allowance of 50% for errors and growth), and four real users joined in. The conclusion was that the system was noticeably slower than when unloaded, but very definitely better than the Data General under heavy load, and always acceptable.

The cut-over to live operation went well, without any significant performance problems, and was also judged a success. In live use there have been a few problems related to a small number of rather extensive and inefficient reports, which are being addressed by making these more efficient. In general, however, the system has performed well, despite eventually being a much lower specification than the hardware manufacturers would have recommended from the simple numbers of users.

7. Conclusions

We have learned a number of lessons from this exercise:

- You need to choose your approach carefully. This prototype-based exercise cost around £20k. Offset against this were notional savings of around £200k against the original proposals for hardware and licensing, and a very real (but less quantifiable) value in reducing the risk to the business and the project by getting it right first time. Thus in our case the costs were well justified. If you don't prototype, you have two real options: over-specify the hardware (which may not cost too much in some smaller systems), or take the best guess from the available documentation, and manage the risk.
- Know what you need to measure, and get good first-hand measurements of all significant aspects of any existing system and your prototype. Do not trust data which you haven't gathered or checked directly - as with our database sizing unforeseen factors can have dramatic effects, particularly when comparing very different architectures.
- Concentrate on the factors which are likely to be a bottleneck in your system. In our case, the disks were likely to be "good enough", but we needed to get the processors right. In a disk-bound system this would have to be reversed.
- Do make sure your prototype covers a range of uses. In hindsight, we should have built reporting operations into ours, and we would then have identified early some aspects which have caused problems, as well as making the cacheing and memory usage more realistic.

- Be prepared to invest in good test tools. Our approach, writing our own, should not be appropriate or necessary in every case. However, you should define the requirements for these tools carefully, and be prepared to be inventive to make the tools provide you with the data you actually need.
- Do get the users involved in the performance tests. However, don't rely on large amounts of manual input - it's difficult to control, and difficult to repeat. An automated test suite will prove to have a number of uses during the development and implementation.

The method we have described worked well in our case. It is not a substitute for a documented analysis of usage, projected growth and required performance but it is a way of deriving them where a previous system exists. Neither does this method remove the requirement for skilled system design, but it will reduce some of the risks if properly used. A performance prototype like ours could be even more useful where the architecture of a system is more complicated.

Over the years, we have both seen systems fail because the sizing and performance testing were not right. In this case, we did get it right, and hopefully this paper will help you to size your system correctly.