

Chapter 2

Performance Theory

Here and in the subsequent chapters as well, we shall deal with systems and applications separately. In practise, however, both aspects are intertwined with each other, and changes in parameters on one level will, with near certainty, have impact on the other. But, there are at least two reasons for reducing the over all problems by separate considerations:

- Separate problem zones often are related to separate key challenges.
- Measures to solve problems can be specific to one level or another (without loosing sight of possible interactions between the two).

A general picture of performance complexities usually presents a problem mix and thus aggravates singling out of critical aspects individually. In this chapter, therefore, we shall look at the two levels—systems and applications—one after the other. Because of their distinct nature, the respective sections are structured differently.

2.1 System Performance

This section concerning system performance is again subdivided into

- hardware parameters,
- operating system parameters.

Again those are in turn intertwined, since both depend on each other. Both aspects constitute what is called system architecture and will in turn again be treated separately, where possible. The selection of operating system parameters available obviously depends on the hardware manufacturer. On the one hand, there exist a nearly infinite combination of operating adjustments with respect to a given hardware configuration of the same manufacturer—only constrained by the real existence of the applications to be supported. This brings us back to the already mentioned boundary conditions. On the other hand, the same version of an operating system can be deployed on different hardware configurations.

Lets look at the hardware first (Fig. 2.1).

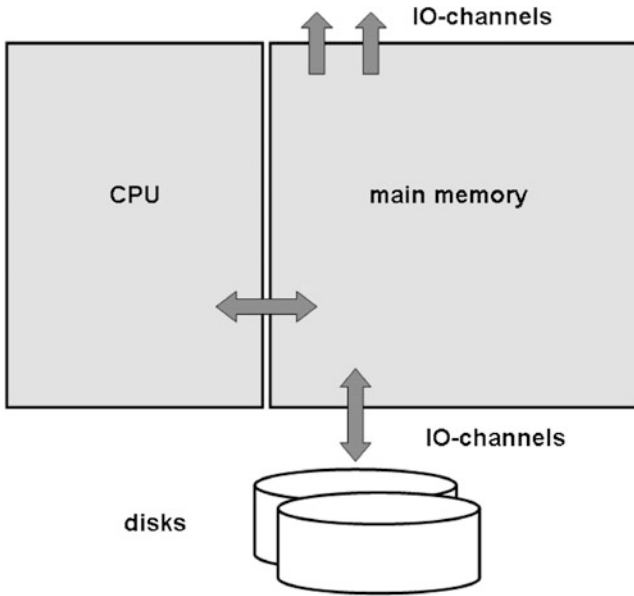


Fig. 2.1 System performance

2.1.1 Hardware Parameters

2.1.1.1 General Remarks

In actual fact, this caption should really read: system or hardware components. The over all system configuration itself can be subdivided into (Fig. 2.1. as 1.2):

- CPU
- main memory
- disk storage
- I/O channels.

Of course there are many more other elements belonging to the hardware domain, such as terminals, modems and communication components. They will not be treated separately here within the context of our performance considerations (such as configuring the cursor speed by mouse programs). The theory of system performance deals with those four elements mentioned above in this sequence. The importance of each component with respect to a specific performance situation depends on the type of application, the number of users and other factors. Therefore, no priorities should be discussed at this stage.

These four components have mutual influence on each other. This will be explained in the course of our considerations. Although the resources mentioned will initially be treated in isolation, their impact with regard to a specific

performance scenario depends—as already mentioned—on the applications themselves, which rely on these resources. To isolate a specific problem, one needs certain evidence to bring basic mechanisms into the open, which have influence on such a specific performance scenario with respect to the application in question.

Before attending to the resources and their specific features in detail, some general principles should be addressed. The following questions are part of the discussion:

- In which way can system performance be tested generally?
- When does measuring performance make sense?

Performance testing is a discipline, by which in the end influence can be exerted over people, processes and technology to avoid risks, which can arise when introducing new systems, upgrades or patch installations. In short, performance tests consist in creating a typical system load before installing new applications—to measure performance, analyse it and collect end user experience. The aim is thus to identify performance problems under production-like conditions and eliminate them beforehand. A well prepared performance test should be able to answer the following questions:

- Are response times satisfactory for end users?
- Can the application cope with the anticipated system load?
- Is the application capable to accommodate the number of transactions expected from the business cases?
- Will an application remain stable under the expected or even unexpected load scenarios?
- Will end users have a positive surprise, when going live, or rather not?

By answering these questions, performance testing helps to control the impact of changes and risks when introducing systems. In this sense, performance tests should comprise the following activities:

- emulation of dozens, hundreds or even thousands end users, which interact with the system,
- consistent repetitions of load sequences,
- measuring response times,
- measuring system components under load,
- analysis and final report.

Tests and analysis should take place before bottleneck situations occur. Otherwise, performance measurements would have to take place later on a running system, including later all relevant measures resulting from the analysis of performance data. This means that for example load distribution over a working day has to be estimated prior to applications going live. The same is true for the decision, which jobs to be launched at what times. Once throughput problems really occur after system introduction, the following sequence of actions makes sense (Fig. 2.2):

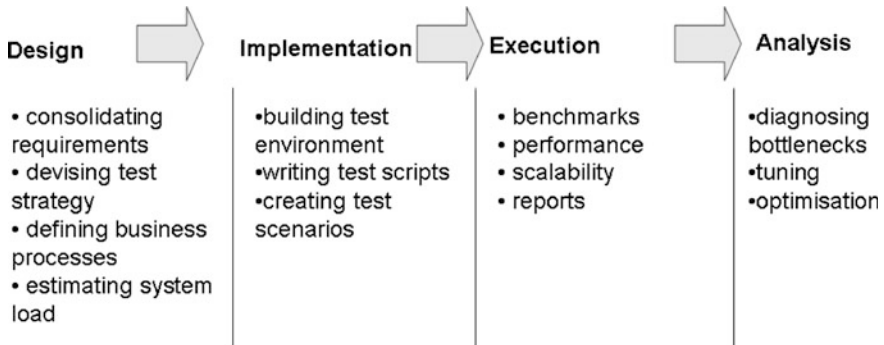


Fig. 2.2 Step by step approach for performance optimisation (HP document 4AA1-4227ENW)

- Initially measure CPU under load,
- thereafter memory usage.

If CPU and memory show no problems, I/Os to disks should be investigated.

Besides preemptive performance measurements discussed so far measurements on production systems should be executed at regular intervals and continuously—especially then, when subjective suspicion about bad performance is reported: long response times for dialogue applications and standard reports needing long execution times.

One problem, however, is the selection of a suitable yardstick. The subjective sentiment of a user sitting in front of a mute screen waiting for the return of his cursor—thus experiencing a pure response time problem—is normally not sufficient to pass for the status of a measured quantity. Generally one would want to examine—besides online functions or GUIs—additional batch runs, the execution times of which can be quantified, to get the overall picture.

Performance tests thus present a tool, with which to find out, whether a system update will lead to improvement or degradation. The conceptual design of such tests in the preparation phase should not be underestimated. It depends on the actual situation on site:

- number of users,
- expected throughput,
- existing hardware, etc.

In any case, it is important not to test isolated applications, but to either look at simulated real conditions or systems as they are found, comprising a significant number of system processes running in parallel.

Concerning individual users performance tests can reduce to tangible key indicators:

- response times and
- dwell time.

These quantities have direct impact on work progress and thus on personal productivity supported by a suitable IT infrastructure. Business process steps accumulate purely operational time and waiting time for system responses or time for system non-availability. Waiting times are a direct consequence of the actual system utilisation in each process step. In this sense, performance tuning has a direct effect on productivity.

Against these demands stand the expectations of operations, emphasising a different direction. Its interest is orientated on maximising throughput on the job level. This means in reality high processor capacity utilisation. Both expectations—those of the end user and those of operations—have to be taken into consideration to arrive at tuning compromises. For this, certain bench marks are available which can be used as a case by case reference. These include not only technical data, but also estimates, which should help to avoid inefficient usage of system resources. Before taking first measures following a performance analysis, the following clarifications should be made:

- Is the general emphasis on throughput only?
- Is the performance of the overall system unsatisfactory?

Quantitatively these questions could be narrowed down to:

- transaction rates depending on response times,
- throughput rates depending on job execution times.

With respect to the user, there are the following key indicators:

- response times belonging to different transaction types,
- execution times for specific jobs that have been launched.

The performance of a complete system can be again be traced to its various components; but also bad production scheduling can be the real reason, when job control is unbalanced. When deploying measuring tools one has to take into account that they themselves will use up some resources.

An important key indicator when measuring is the so-called system time. It is the sum of the following items:

- controlling time sharing between several parallel processes,
- controlling I/Os,
- controlling swapping for main memory usage.

Experience shows the following benchmarks for a typical CPU:

- overall system time: 10–20 %,
- timesharing: 5–10 %,
- I/Os: 2–6 %,
- swapping: 1–2 %.

Figures 2.3 and 2.4 give an indication of the essential approach and sequences of a tuning process.

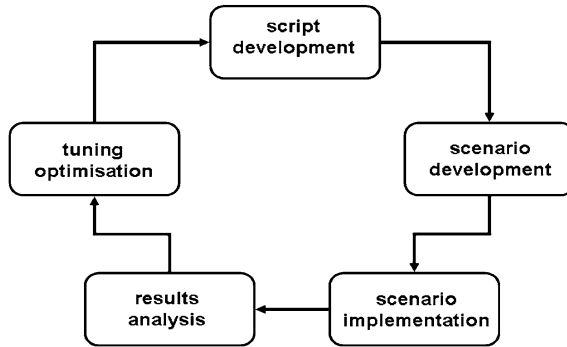


Fig. 2.3 Optimisation cycle (HP document 4AA1-5197ENW)

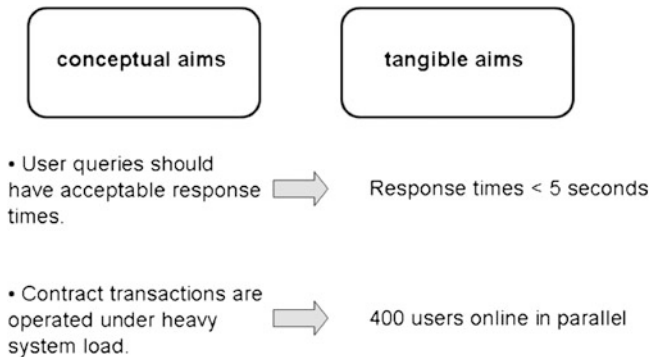


Fig. 2.4 From concept to objectives

At the end of these introductory remarks, two more general observations: upgrading existing hardware is often not the panacea for performance bottlenecks, as will be shown further on. And performance problems grow exponentially with the data volume to be handled. After these basic considerations, the individual resources in question will now be examined separately.

2.1.2 CPU

CPU stands for central processing unit. In the context of performance, CPU investigations focus on power, that is, throughput through the processor. This power is measured in million instructions per second (mips). This, however, stands on paper only, which tells us something about the useable power and about overheads and task management without I/Os or waiting queue management for other resources not being available or about virtual memory management for that matter.

Table 2.1 RPF determination

CPU	RPF batch	RPF online	RPF wheighted

CPU power is a critical factor for those applications, which are strongly CPU bound, such as scientific and technical applications with long sequences of mathematical calculations.

One possible yardstick can be the “relative performance factor” (RPF) (see Table 2.1). This quantity gives some information about the power potential of a CPU with respect to production conditions as realistically as possible. For its determination, both online applications and batch processing are taken into account. For online, the transaction rate is taken as a measure, and for the latter, the number of jobs executed per unit time. These measures are weighed in accordance to their relative importance in a given application setting: 20 % batch, 80 % online for example. But a single number alone is not capable to provide a definitive assessment about over all CPU behaviour. Normally, the application landscape is too complex for that. Other factors of influence are given by the server architecture. All these considerations, however, do only make sense, when CPU performance and main memory capacity stand within some reasonable relationship to each other.

A CPU can be in the following states:

- user busy
- overhead management
- waiting
- idle.

“User busy” refers to the execution of application tasks; “overhead management” indicates that the CPU is busy with managing itself, while going through waiting queues or redistributing priorities for example. “Waiting” points to the fact that a required resource is not available, and “idle” means that there are no demands on the CPU at present. Overheads can be broken down as follows:

- memory management/paging,
- process interrupt control,
- cache management.

Table 2.2 Increase in management overheads in a multiple CPU environment (Siemens: BS2000/OSD performance-handbuch Nov. 2009)

No. CPUs	Increase CPU time (%)
2	5–10
4	10–15
6	15–20
8	20–30
10	25–35
12	30–40
15	35–45

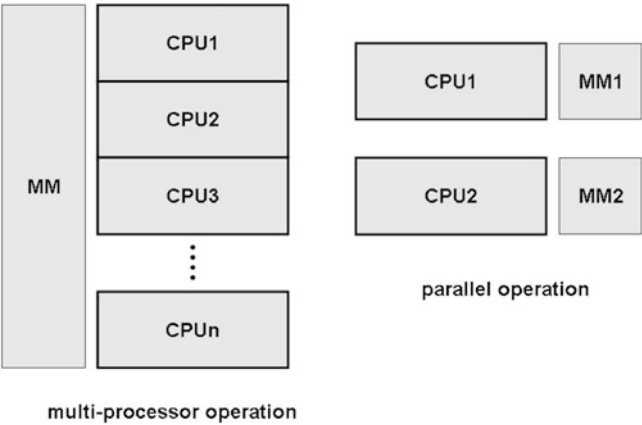


Fig. 2.5 Possible multiCPU configurations

These problems look entirely different in a multiprocessor environment (Table 2.2). This environment is constituted from central processing units on the one hand and I/O processors on the other. To optimally exploit such a configuration, applications have to be written accordingly. This includes allowing for parallel processing without mutual interferences.

Such configurations have to be distinguished from others, which utilise parallel processors for resilience purposes to guarantee continuous availability of systems (Fig. 2.5).

On the task level, a specific task can only be assigned to one particular processor and vice versa. This means of course that upgrading hardware by adding more processors has a limited impact to solve performance problems concerning response times. In the limit, only a quicker CPU assignment takes place without any other throughput improvement. At the same time, a significant acceleration of batch jobs can only be achieved by massive parallelism. Besides, one has to take care that by increasing the number of processors demand on synchronisation with main memory increases as well producing additional overheads. Table 2.3 shows as an example improvement potentials for the execution of independent tasks by adding more CPUs.

Table 2.3 Improvement effect after adding CPUs (Siemens: BS2000/OSD performance-handbuch Nov. 2009)

No. CPUs	Factor
2	1.7–1.9
4	3.3–3.6
6	4.6–5.2
8	5.8–6.6
10	6.8–7.8
12	7.8–8.8
15	8.8–9.9

Table 2.4 Load as a function of the number of CPUs (Siemens: BS2000/OSD performance-handbuch Nov. 2009)

Load (%)	No. CPUs
75	2
80	4
85	6
90	8

Such upgrades generally mean that memory and disk drives will have to be adjusted upwardly as well.

Many manufacturers recommend reference levels for optimal utilisation of their CPUs. In the following, we shall concern ourselves with mono processors only. For online-oriented systems, CPU utilisation should not exceed 70 % for main application modules. In excess of that value, management problems concerning waiting queues can occur with the consequence of longer wait times. Naturally, possible utilisation figures exceed these reference levels for a multiprocessor environment (Table 2.4).

To control CPU tasks, certain routines are employed, which manage and supervise. The following parameters are important for task management:

- activation,
- initialisation,
- multiprogramming level,
- priority,
- multiprogramming level (MPL) per category,
- resource utilisation (CPU), main memory, paging activity,
- system utilities (CPU time, number of I/Os),
- assignment of access rights for main memory usage,
- assignment of CPU resources,
- state “active, ready”,
- de-initialisation,
- resource usage at waiting times,
- state “active, not ready”,
- expulsion.

Since a CPU is used by many tasks at the same time, utilisation is an important criterion for waiting times. The average service time by the CPU is, however, small in comparison with input and output times. This means that response times are influenced mainly by I/O processes, especially when reading or writing on storage media. The 70 % utilisation is determined solely by those. On top of this, some manufacturers permit manual priority allocations, such that tasks that had been allotted low priority initially can now jump waiting queues. 100 % utilisation is an ideal value, which will never be obtained in practice.

Priorities refer to access to CPU resources, I/Os are not taken into account by this. Normally, priorities are assigned in such a way that online tasks get higher priority with respect to batches. System programs on the other hand have even

higher priorities than all other online functions. Within program processing, the system itself calculates priorities according to a particular algorithm, taking into account waiting queue positions, swap rate and other parameters. Allocation of an external fixed priority is possible, should, however, be handled with care, since this priority will not be adjusted dynamically thereafter and could lead to the expulsion of other processes. In this way, a few users, which deploy large programs, can be assigned low priorities against many users having to attend to only small tasks. At short notice, urgent business function could be assigned higher priorities as well as memory intensive programs blocking everything else, to liberate the system.

When talking about high CPU utilisation, the figure is around 90 %. This does not necessarily mean a bottleneck situation. At high utilisation, however, it is important to watch the efficiency of CPU operations. The following measures should be observed:

- checking the level of the I/O rate,
- checking the level of the paging rate.

One measure for the efficiency of a given CPU at high utilisation is the ratio of waiting time against CPU/hardware service time.

When longer waiting times occur, this may be due to the following causes:

- non-optimal adjustment of system parameters,
- the over all CPU time requirements for all tasks are too high. An analysis of waiting queues could lead to the origins of this situation.

An important role concerning CPU management is played by the handling of interrupts. Obviously, swapping means the interruption of executing code. By this, a program or a part of a program is stopped, the code segment swapped out of main memory, and the task is directed into a waiting queue, from where it will gain access to resources later according to its priority (normally position in queue). Interrupts can have the following causes:

- priority-triggered resource access by the operating system,
- termination of a self-contained part of a program,
- technical error (“exception”),
- a longer pause when addressing a program page,
- I/Os,
- lacking data,
- status “idle” with time-out.

But even at low CPU utilisation levels, performance problems can occur. These can have the following reasons:

- I/O conflicts,
- high access rate to storage media by many users at the same time (central data files),
- paging rate too high,
- bottlenecks due to server tasks,

- inefficient parallel processing for certain programs,
- CPU time for the execution of certain programs,
- too many system calls,
- system self management,
- too many library routines,
- too many I/O routines,
- too many users in parallel,
- multiprocessor management,
- unfavourable CPU clock cycle per process,
- processor deadlock,
- sequence of machine instructions.

If performance problems continue to persist, even when CPU is “idle”, these problems are to be looked for elsewhere. By upgrading the CPU nothing will be gained.

2.1.3 Main Memory

One significant characteristic of main memory can be found in its nominal size, that is, the number of bytes available for central processing and its temporary storage for this purpose. The absolute size of main memory with respect to a performance scenario can be

- just right,
- too big or
- too small.

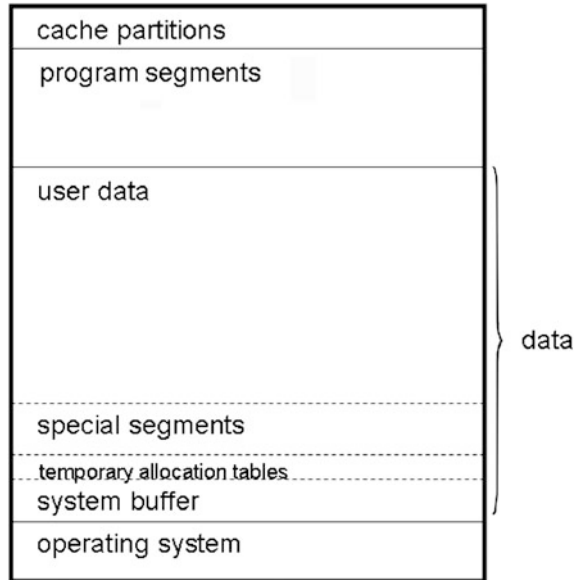
This sounds trivial. But the following has to be taken into consideration: an oversized memory, often recommended to compensate for bottlenecks, can lead to secondary CPU problems, when processing power does not follow in step, or will expose the limits of I/O channels. The reason for this is the number of possible tasks, which can be loaded and will exceed the capabilities of these two other resources in such a way that waiting queues will be generated.

Of course, an undersized main memory is the most obvious problem. Main memory has to deal with (Fig. 2.6).

- Program segments,
- data,
- cache partitions and
- parts of the operating system.

Normally, some part of main memory will constantly be occupied by some basic functions of the operating system: base memory as a fixed portion of the overall storage space, 10 % for example. Other portions of the operating system will be loaded later, when the need arises. An important portion will be needed for

Fig. 2.6 Main memory usage



user processes. This is the reason, why special care has to be taken, when updating the operating system with a new version, which may include additional functions. Reconfiguring has to take this into account.

Cache partitions are needed for storage segments containing data, which are addressed repeatedly. These partitions can be configured but also be assigned dynamically. The latter does not present a particular problem for the cache, even when bottlenecks may lead to restrictions concerning the cache portion and part of it is outsourced to a disk. Performance then degrades by the fact that some cache data now have to be read directly from the disk. On the other hand, intensive cache usage leads to interrupts and address mapping. This in turn generates additional CPU overheads.

Data occupancy of main memory has the following sources:

- user data,
- temporary addressing tables,
- special segments,
- system buffers.

User data occupy the address space, which has been assigned to a specific user for sorts for example. Special segments are used for messaging or communication buffers and system buffers for certain system tasks. The rest is available for program segments or for paging.

An important factor when judging main memory performance is the paging rate. Paging rate is defined by the number of instances per unit time, when actually memory-resident code pages being used for the execution of a program by the

CPU are temporarily removed, replaced by others and the original pages later again be reloaded. This sequence is called swapping.

There are some algorithms identifying those main memory entries, which are earmarked for swapping. Normally, the resource waiting queues are polled by each system cycle within the frequency of the system clock to find out which segments have not been addressed during the previous cycle. Those will be flagged as overlay candidates. During the next cycle, the overlay candidates are again checked and possibly swapped, if memory space is needed. If there are many overlay candidates and long dwell times for certain segments, which could be swapped without process interruption, only marginal memory management overheads will occur.

High paging rates can be critical for an efficient use of the main memory resource (Fig. 2.7). Reasons for this are as follows:

- too many processes,
- programs with a too large size.

Therefore, manufacturers release recommended maximum values. Memory management for a smoothly performing CPU should not account for more than a few percentage points. Otherwise paging rates, which are too high, will lead to bad response times, because CPU resources will be used for that. Background: in both cases mentioned above, the CPU first tries to reorganise main memory occupation by creating a concatenated address space (generating considerable overheads). If this does not work, the CPU gives up, that is, processes of low priority will be interrupted, swapping takes place into virtual memory or into the paging areas on the disks. Consequences are as follows:

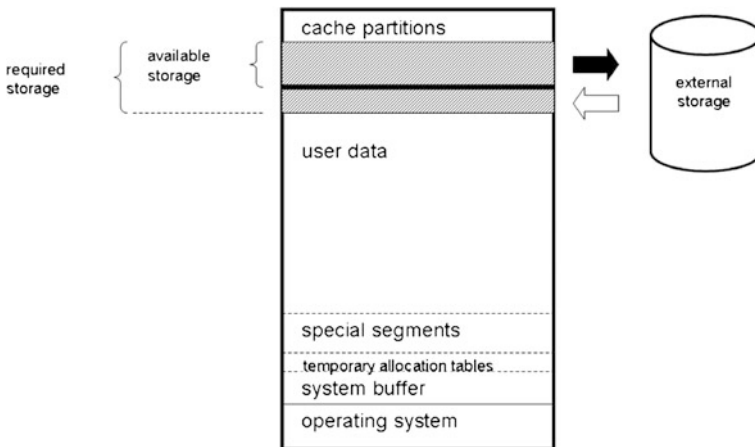


Fig. 2.7 Paging

- lost CPU resources for memory management (increase in subjective response times),
- interrupted user processes (increase in subjective response times).

In case these problems become chronically, there is no way around an upgrade of main memory. One reason for this is that external swapping areas have to be specially configured. If they are too small, paging becomes a secondary bottleneck with additional I/O waiting times and corresponding memory management.

To avoid main memory bottlenecks, it makes sense to regularly and prophylactically take measurements with a suitable monitor. The following values should be captured:

- the number of pageable pages (size of main memory minus the number of pages in permanent residence necessary for system purposes),
- the number of globally used pages (number of pageable pages minus the number of freely usable pages).

Similarly as for the CPU, there are reference values for main memory. Usage of up to 75 % is supposed to be non-critical. By usage of continuously more than 90 % an upgrade is unavoidable. As is the case for CPUs, there is a task management with more or less similar triggers via control functions for main memory management including a page administration algorithm:

- activation
- deactivation
- forced deactivation
- expulsion.

Summarising, the following scenarios can happen:

Memory demand is higher than available. This creates a paging problem. This means that certain process sequences will be relocated temporarily onto decentralised storage media leading to noticeable performance degradation. Solutions can be gained either by hardware upgrades or application optimisation. On the organisational level, the number of users could theoretically be reduced, which is generally not regarded as being acceptable.

The opposite scenario is given, when sufficient memory is available. There is no third option between these two states. In this context, a specific program does not have to be loaded completely for the execution of certain functions. Program pages are loaded in accordance with their address sequences one by one leading in turn to swap-ins and swap-outs—thus paging processes. In any case, statistics show that most programs remain during 80 % of their execution time within only 20 % of their own code. Inactive processes are mainly caused by online applications—when users remain in front of their screens and do not enter anything, while the program itself remains open via the GUI. To optimise with respect to paging, the following measures can be applied:

- usage of common subroutines by several programs at the same time,
- code size optimisation,
- usage of common libraries,
- memory configuration with respect to the estimated number of users.

2.1.4 Disks

There are some simple facts that explain, why absolute disk space influences system performance: should totally available disk space be occupied by a theoretical 100 %, the system will grind to a halt. Files can no longer be created (even temporary or spool files) or even updated. But, of course, there are other problems, which will be noticed much earlier.

Every system coming close to 90 % disk space occupation will indicate performance problems regarding response times as well as anything as a smoothly running production environment. For problem-free operations, one needs free storage space for

- backups
- end of the month results or
- database reorganisation.

These are the reasons, why disk occupation should not exceed 75 %.

Normally users and administrators have only limited influence on the physical localisation of files on external storage media (volumes). One has to keep in mind that one physical volume may contain several logical volumes. Volumes are addressed via an identification number. Access times depend on the type of access technology, management of storage space, but also on the fact, whether a configuration uses several computers at the same time. Other impairments can be created due to parallel disk mirroring, especially in connection with the RAID concept. Redundant arrays of independent disks (RAID) function the other way round by grouping several physical volumes under one logical one. In this way, a corresponding data redundancy is created.

Manufacturers offer modules for device management with the following functionalities:

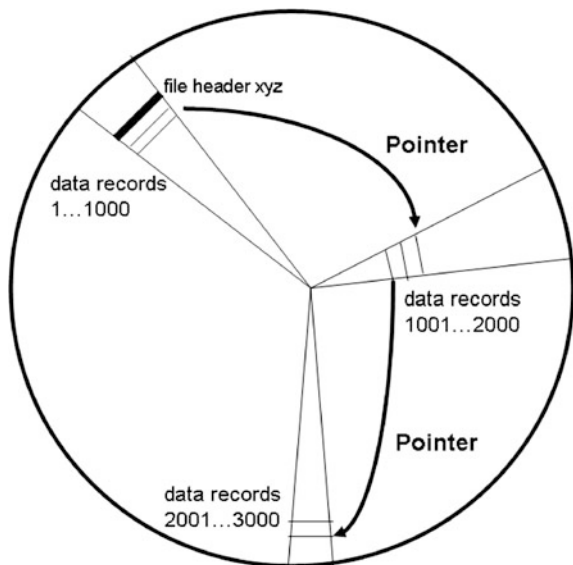
- monitoring of disk occupancy,
- reservation of disks,
- configuration,
- surveillance of access readiness,
- partitioning to optimise access to logically connected tables.

For all those reasons, it is important to identify and plan for storage locations prior to new installations, determine where which data should be placed and design an appropriate schema before effective initial loading.

Compromises need to be found between the objectives of throughput optimisation of an individual process against the overall throughput for the whole disk. To measure a single process, a large test file is required. For the overall throughput of a single disk, this is more difficult. Actually, this can only be achieved by a simulation covering all processes. Thereafter, measures can be taken respecting the different interests of applications. Another compromise concerns the optimal usage of the storage medium with respect to throughput. In this case, one has to distinguish: large files having little storage efficiency should be placed on single defined disk, many small files on others.

Another aspect concerns the fragmentation of information (Fig. 2.8). It is indeed not the case that all data records belonging to a specific table should reside concatenated on a particular storage medium—not even initially. Files and data records are fractionated or split and distributed according to an internal algorithm over a disk together with corresponding pointers referring to respective subsequent addresses. During the life cycle of an application, this state of affairs degrades even more since files are updated by deletions or additions of data records. By this, gaps on the storage medium are created; then, these gaps are refilled. These new creations do not necessary belong to the same file nor are they in a specific logical sequence. In the end, search processes for queries will be prolonged and thus response times. From time to time, a defragmentation should be run. This results in gaps being filled up and associated data being brought closer to each other.

Fig. 2.8 Fragmentation of a hard disk



2.1.5 I/O

As already mentioned, there is a danger that a significant portion of CPU resources is required for the management of I/O waits. This is especially the case, when they are caused by process interruptions. Besides disk access themselves, important roles are played by the number and speed of communication channels and their controllers. Those are also required for terminals, modems and other communication hardware. A too small number of I/O controllers leads to communication bottlenecks. But on the other hand, a given CPU can only manage a limited number of controllers itself.

There are strong dependencies between the performance of external disk drives and the capacity of communication channels. Some manufacturers provide control mechanisms as part of their data management systems for input/output processes. The reaction time for input/output commands is generally one order of magnitude longer than that for CPU commands.

To begin with our interest shall centre on input/output to and from external storage media (Fig. 2.9). In programs, those calls are realised via intrinsics or I/O program calls (get, put, etc). SQL queries (read, write) are translated into equivalent execution instructions. Of importance for the execution of such or similar queries is the structure of the databases in question. The duration of search operations differs for sequential files, index sequential files or relational database management systems. Furthermore, execution times depend on the type of access. Reading access is obviously faster than updates, where a record has to be found first, then read and finally written to.

Once input/output is needed for the continuous execution of a program, this part of the code is put into waiting until the input/output sequence is completed. Thereafter the program continues to run. From the point of view of the CPU, the program rests in a waiting queue, which is controlled by a priority algorithm. This queue contains also parts of other competing programs, which means that the immediate execution of the program in question after the I/O is completed is not guaranteed, but will proceed only according to the actual internal priorities. In case the input/output happens via specific channels or special devices, there are again waiting queues, since other users demand access to those resources as well.

Input/output is thus a complex process, which succeeds in combination with various waiting queues and data buffers, into which the requested data records are read.

The so-called data cache helps to accelerate accesses (Fig. 2.10). Data cache is a buffer, which stores those data requested last, while the user or the program is occupied with different things. Now, is the last request for some reason repeated, this information is made available immediately from the cache. On top of these data, cache provides for a minimisation of disk accesses and the bundling of write operations (write behind method). Data caches can be configured. Data throughput is influenced by their size. Compromises have to be found between cache reservation in main memory, I/Os and CPU load for swapping purposes.

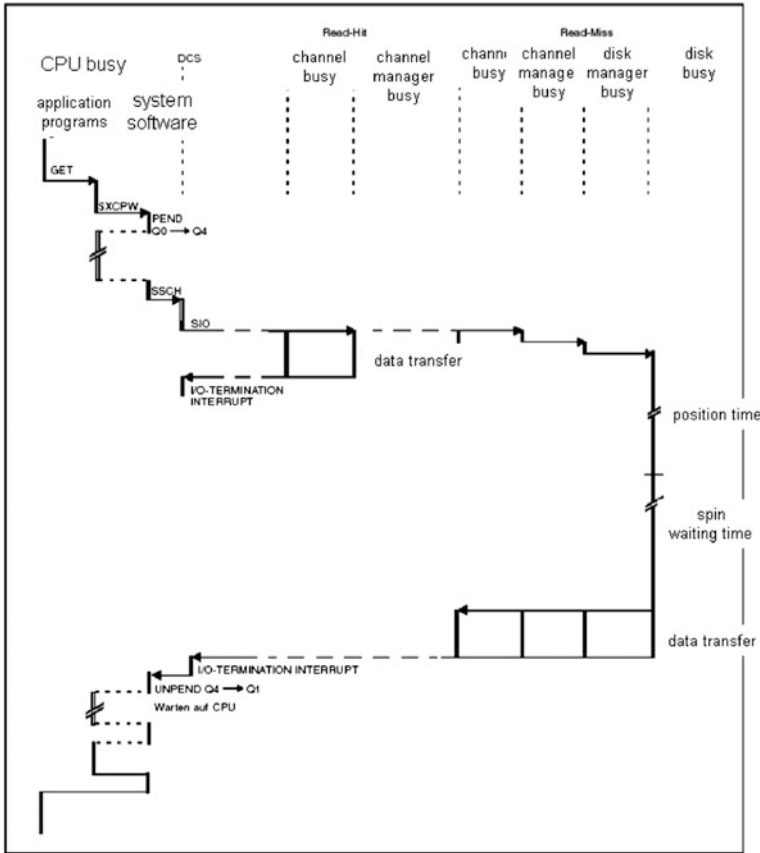


Fig. 2.9 Disk I/Os over time (Siemens BS 2000/OSD performance-handbuch Nov. 2009)

I/O time depends in turn on the technology of the system bus and the channel structure for the I/Os. In case of several external disks, there have to be compromises between the total transfer rate across all disks and the individual transfer rate of single disks. Maximising the overall transfer rate impedes the individual transfer rates. To optimise, one can split applications in such a way that certain disks contain large files and others in turn many small files for less important I/O demands.

The optimum solution obviously is an even distribution of disk I/Os across all disks. Each disk should have its own controller. Shared controllers with several channels on the same bus should be avoided. To sum it up, transfer rates are thus dependend on a combination of disks, bus performance and controllers. Optimisation, however, does not necessarily succeed by making all components faster, since new bottlenecks can arise by a good performance of one component to the detriment of one of the others.

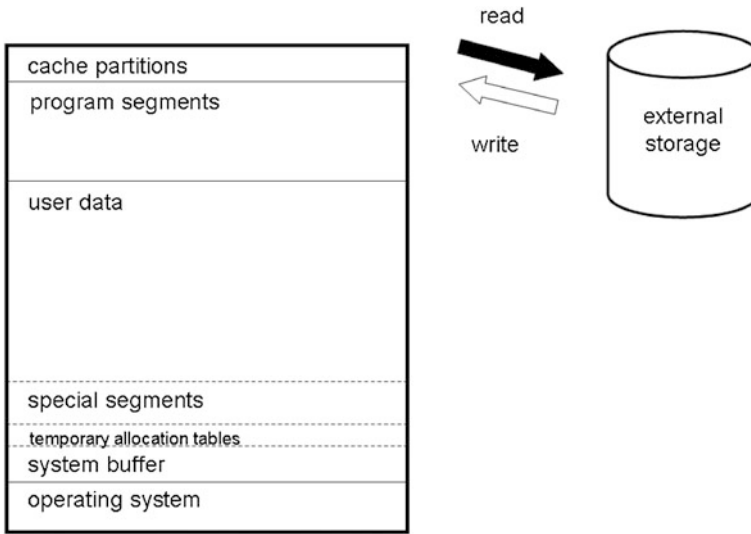


Fig. 2.10 Cache management

Performance data of disks do in turn depend on the positioning time (time to move the disk head from one data track to the next) in dependence of data contiguity. In case of high fragmentation and data distribution across many applications, frequent jumps of the head will be the consequence. This in turn leads to accelerations and decelerations. For this scenario, search operations themselves are much longer than reading or writing on their own. To improve this situation, the various possibilities for partitioning provided by the manufacturer should be put into use.

2.1.6 Operating System Parameters

Every operating system carries besides its capabilities to support applications also its proper functions, which influence and demand system resources. These comprise the following:

- program calls,
- certain commands,
- system control functions,
- configuration parameters,
- system tables,
- functional sub-systems: utilities.

Certain operating system commands can influence performance. Some of them are used for:

- communications control,
- job limits,
- logging,
- memory allocation,
- priority assignment.

Directly influencing performance are the following system control functions and configuration parameters:

- communication settings,
- job limits,
- logging,
- memory allocation,
- priority scheduling,
- settings relevant for data systems,
- spool parameters,
- time outs,
- cache sizes,
- system tables.

System tables need special attention. Generally, it is understood that table overflows will bring the system down. On the other hand, oversized tables will occupy too much main memory. Concerning utilities, one has to decide on the organisational level which circumstances will allow to provide certain users or applications with which utilities. These utilities may include:

- editors,
- system monitoring routines,
- debugging software,
- format handlers,
- compilers,
- database handlers.

Besides all these, it is important to check, whether new versions of operating systems contain features, which can positively be employed to improve performance. This should be an ongoing process.

2.2 Application Performance

2.2.1 Application Parameters

Although performance tuning on the basis of application parameters is usually time consuming and will normally not show visible results in the short run, it nevertheless is often without alternative to gain success in the medium and long term. These are some influencing factors:

- programming language,
- program architecture,
- number of subroutines and external calls,
- data management,
- I/O procedures,
- GUI or batch processing,
- size of code segments,
- communication processes with other applications.

If it can be chosen, the programming language plays an important role for the future performance of an application. This has to be put in relation to realisation times for a modular program architecture against the classical structured programs, which has lost in importance with the advent of modern programming languages. In a modular architecture, application modules are decoupled from each other although united under a common front end. This relieves main memory. Besides, certain modules from different parts of an application can be addressed by different programs or other program parts and in this way utilised in common—for example, error routines.

Although one should expect that every new release will deliver acceptable application performance, the results depend strongly on preparatory measures. They should take into account demands of the users and their performance expectations. The basic process sequences, which IT is going to support has to be taken into account. In the end, the following questions should be answered:

- Has the application architecture to be adapted to satisfy certain performance criteria?
- Will an upgrade of the IT infrastructure be required?
- Is performance degraded by GUI design?
- How will system load be affected by an expected increase of throughput?

Occasionally, some factors can be tuned with small effort—like I/O procedures or a switch from GUI to batch. Others demand higher investments. The transposition from one programming language to another or changes to the data model belong to the latter.

In addition to measured quantities, which are of importance to system performance itself, there are a number of characteristics with direct relevance to applications. Task specific demands on resources over time and at peak hours belong to these. They relate to the following:

- CPU time,
- I/O results including accesses to certain data tables and disk access statistics,
- waiting times,
- paging rate,
- channel transports including TCP/IP connections,
- possible communication events within a cluster configuration,
- accesses to main memory,

- response time statistics,
- cache hit rate,
- number of job executions.

This information can be gathered either by measuring monitors or partly from log files once the relevant transaction types have been flagged for recording. To record transactions, log files need to be adapted specifically with respect to two criteria:

- formatting
- selection.

Formatting means, that care has to be taken, that entries are disburdened from technical information. This requires that everything with relevance to transaction codes or data encryption, format declarations, etc. should be spared. Only readable text information should be deposited, which give an immediate indication to the analyst about what is happening with a particular transaction.

Only such transactions should be earmarked for recording that are of relevance to a later analysis. Negligible functions like “show date”, “return to main menu”, etc. should be suppressed.

Furthermore, averages and peak recordings over time for a typical working day are of interest. These will show time dependent bottlenecks, which could initially be decoupled by organisational measures without major technical efforts. Concerning the number of users behind resource utilisation, one has to differentiate between active ones and those, which are only logged in and remain idle otherwise.

2.2.2 Data Management Concepts

Data management is a vast area and presents a high potential for application tuning. All important computers are equipped with data management features in one way or another either as part of the operating system or as separate tools with possibilities to assign logical names to files and to specify a physical storage medium. The system or the tool organises the physical space and writes data to it or reads data from it according to the user instruction in question.

The commonly used expression “data base” in the widest sense means a collection of data records connected between each other with maximum coherence, stored in calculated redundancy and structured in such a way that different interests of many users can be satisfied (Fig. 2.11). In the following, different data management systems will be touched upon. It is unavoidable that historical structures, which do not represent the state of the art of file management today, will get their mention too (Figs. 2.12 and 2.13). Reason is that even of today in many organisations, traditional file handling is still in place, which has not been replaced by modern methods and often is a common cause for real performance problems.

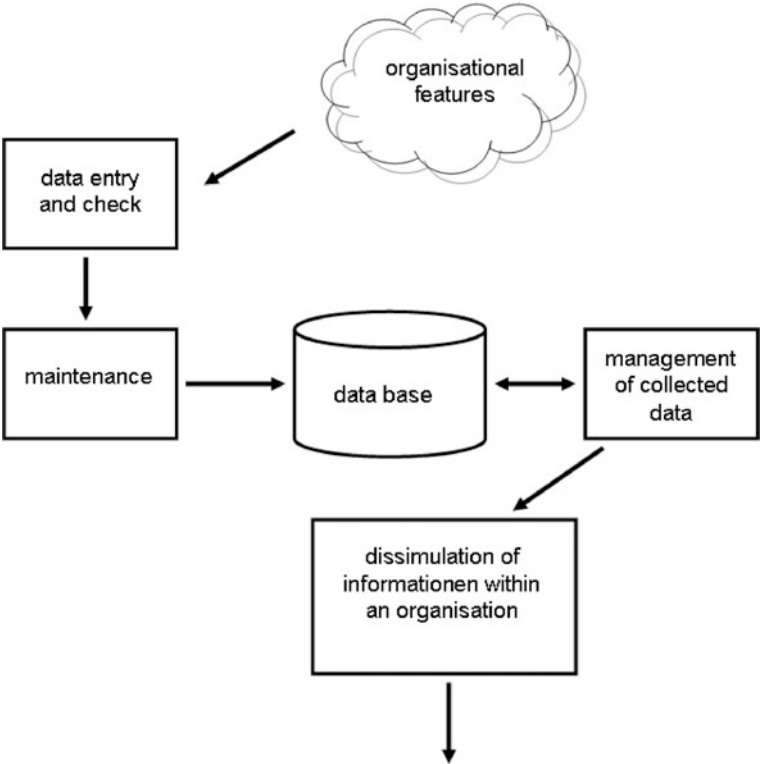


Fig. 2.11 Data base within an organisation

Fig. 2.12 Sequential file

data record1
data record2
data record3
data record4
•
•
•
•
•

2.2.2.1 Technical Conditions

In general, a data management system appears to a user as a software interface, which separates him from the operating system and at the same time from external

Fig. 2.13 Index sequential file

index1	data record1
index2	data record2
index3	data record3
index4	data record4
	.
	.
	.
	.
	.

storage hardware with respect to each access to a centrally controlled and integrated collection of data, which is shared between several applications and is called “data base”. The system provides tools to define the physical structure of the data base and its logical connections, to load data and modify them, to protect these data against incidental damage or unauthorised access and to cater for efficient data queries. A data management system is called “generalised”, if it is driven by a single user-oriented command language for all these different functionalities applicable to every new data base and independent from its internal organisation so to avoid the necessity to each time have to write new data management programs for every new data base.

But even in such a comfortable environment, there remain two unresolved problems:

- the optimisation of data mapping,
- the best possible usage of advanced database management functions.

Already in 1971, standards for database management system (DBMS) were developed by the CODASYL DataBase Task Group—at that time specifically for COBOL as main host language. They proposed a network structure to model relations between data records within a common data base (Fig. 2.14). Hierarchical relations were dealt with as being simply special cases in a chain of network files.

This widely used logical structure can be found in the cross linkage of data records, which organise items within that record hierarchically or in a tree structure. In most cases, most of these systems are capable of mapping a multilevel hierarchy, in which sub-items may consist in turn of other sub-items and so forth.

Purely logical networks are based upon the concept of interconnected files, where each one may be allocated an owner or master record and one or more member or slave records. The higher efficiency of networks against hierarchical structures is given by the fact that one single-record type may be associated with many others.

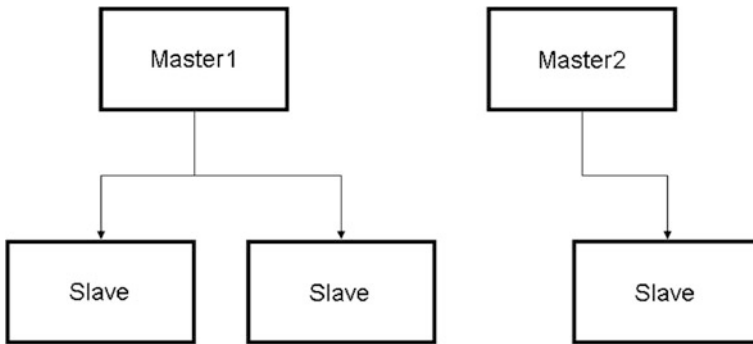


Fig. 2.14 Classical DBMS

A typical DBMS comprises the following components (Fig. 2.15):

- data description compiler,
- data dictionary reports,
- data manipulation language,
- system modules,
- utility programs.

Soon after its introduction, the so-called relational model exerted great influence on data management as such (Fig. 2.16). Contrary to the models discussed so far, the relational model can be defined by simple rigorous mathematical concepts. Furthermore, its underlying structure permits simple queries, modifications, updates and restructuring of the database.

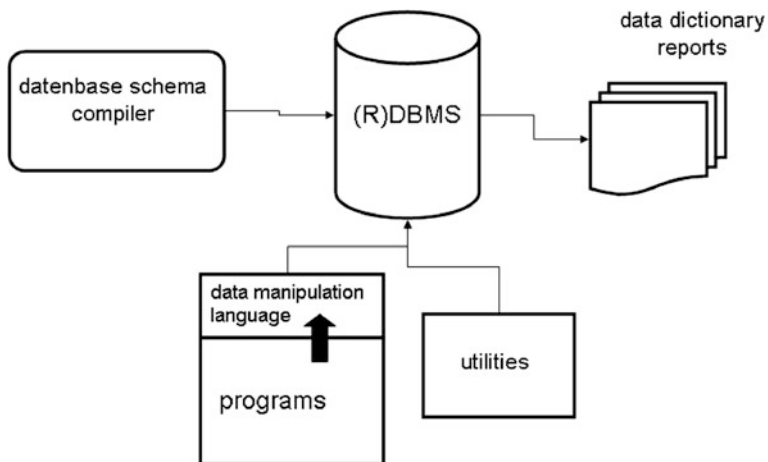
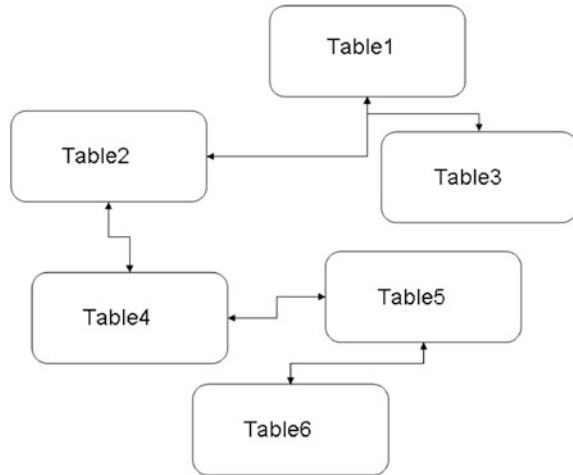


Fig. 2.15 DBMS tool box

Fig. 2.16 Relational database management system (RDBMS)



Altogether, the relational model is more user-oriented. It views a data base as a collection of n possible relations and homogeneous tables, in which each line corresponds to a single-data record, comprising n items appearing only once. When defining relations consistency and lack of redundancy can be guaranteed, once a set of formal rules is respected.

These five basic operations in relational algebra are as follows:

- selection
- projection
- product
- aggregation and
- difference.

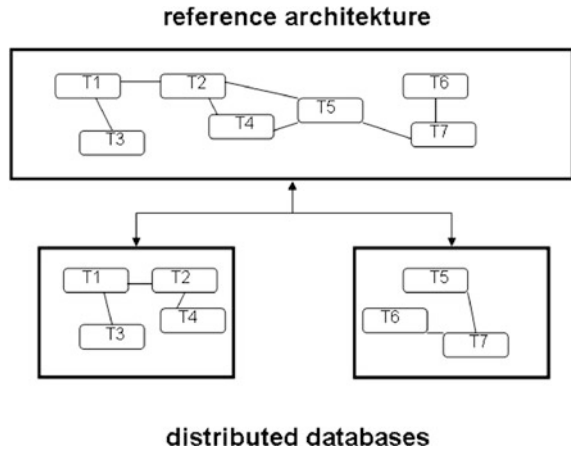
The next logical step in the direction of new database technologies was the development of distributed databases. From a user point of view, its principles seem relatively straight forward and transparent: applications have local characteristics on a local platform. Within a communications network, database functions coordinate access to distributed databases and pretend to a user that the data are locally available. The knowledge about the real data distribution belongs to the DBMS alone. Changes in data distribution do not entail changes in programs.

To achieve this, the DBMS relies on a so-called reference architecture, comprising global and local schemata and the physical data assignment (Fig. 2.17).

2.2.2.2 Accesses

Independent from data management systems in operation, various operating systems offer data access modalities having impact on overall performance. Those comprise the following:

Fig. 2.17 Reference architecture



- buffered or non-buffered access,
- access with or without I/O wait.

In certain cases, both options permit the continuing execution of programs without I/O processes having necessarily been completed. Besides these, table sizes as such and the overall blocking philosophy play important roles, of course. Locking should be implemented on the lowest possible level (item), if feasible, never on table level and—if possible—also not on record level.

2.2.3 Application Environment

Application environment is more than just the GUI visible to the user and the corresponding software and data base, which supports his transactions—including batch. To understand the behaviour of the application properly, one has to look at the interplay between system components and the application itself. This means that it is always a matter of symbiosis between these two aspects. To describe the overall system, one has to take into account:

- the system nucleus: the operating system with its administration of the system resources on hand,
- job management,
- data management at operating system level,
- I/O control,
- other external resources (storage, modems, etc.).

Under certain circumstances, an application comprises also sub-systems and utilities like EXCEL© Sheets, directories and other elements. When setting up an application, one has to take care about additional variables like the following:

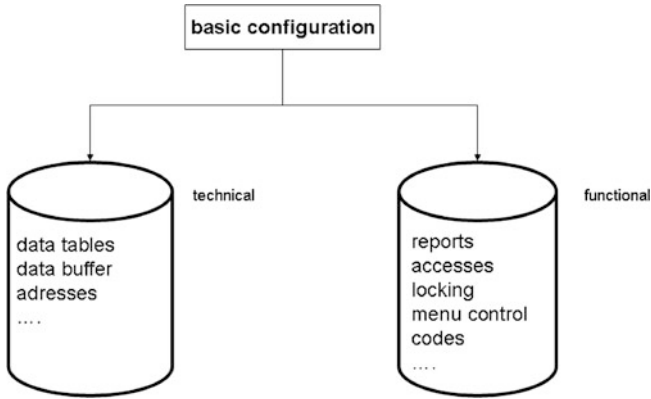


Fig. 2.18 Application world

- application specific buffers and address space,
- authorisations,
- priorities,
- other operating parameters.

All these adjustments influence all over performance, especially the configuration of buffers and the address space allocation in main memory. This means that compromises concerning competing system requirements are inevitable. Figure 2.18 illustrates schematically the environment, in which applications operate.

For an application environment to go live, here are the essentials:

- the actual software release,
- the database,
- additional utilities,
- documentation,
- training,
- authorisations.



<http://www.springer.com/978-3-642-29970-4>

Computer Performance Optimization

Systems - Applications - Processes

Osterhage, W.W.

2013, X, 165 p., Hardcover

ISBN: 978-3-642-29970-4