Malaysian Journal of Computer Science, Vol. 9 No. 1, June 1996, pp. 25-31

RESPONSE TIME CONSIDERATIONS IN REALTIME SOFTWARE DESIGN

Palaniappan Sellappan
Faculty of Economics & Administration
University of Malaya
50603 Kuala Lumpur
Malaysia
Fax: 603-7567252
Tel: 603-7593693
email: sella@fsktm.um.edu.my

ABSTRACT

Identifies several factors that affect the design of realtime software and then discusses some approaches that a software developer may use to meet the inherent timing requirements of realtime software.

Keywords: ***distributed processing, realtime software, response time, software design***

1.0 INTRODUCTION

There is a growing demand for distributed realtime applications in today's business environment. Users must interact with computers to obtain up-to-date information (e.g., flight information, customer inquiry, courier services). Many applications today also require computers to monitor and control devices (e.g., traffic lights, hospital and manufacturing equipments). The software requirements for such time-critical realtime applications are often stringent. Therefore, they must be designed as efficiently as possible.

A realtime software must be responsive to the timing needs of a client's application. It must not only satisfy the functional requirements of the application but also the performance requirements, especially the response time requirement. A realtime software is considered defective, and therefore unacceptable, if it passes all other requirements (e.g., logic, accuracy, human interface) but fails to meet the client's response time requirement [3, 4, 5, 6, 7, 8]. That means, software developers must make a concerted effort to incorporate features that will satisfy the client's timing requirement (in addition to all other requirements). This paper lists some of the factors that influence response time and discusses some approaches/techniques that a software developer can use to meet the application's (client's) timing requirements. (It does not address other user requirements such as accuracy, completeness, human interface, logic, and design structure which must also be satisfied).

2.0 DEFINITION OF RESPONSE TIME

For the purpose of this paper, we will define response time as the elapsed time between the submission of a request by user (human, software, or device) and the system's response to that request. Note that this definition includes the elapsed time between a software's or device's submission (transmission) of data or transaction record (in the form of signals) and the system's response to that request.

3.0 FACTORS AFFECTING RESPONSE TIME

There are many (often interacting) components (factors) that affect the response time of a system. Some of these factors are hardware, software, methods, tools, network structure and the system environment [2,5,7,10,12,13]. A software developer must take these and other factors into consideration when designing a realtime software. Depending on the client (customer), some of these factors may act as constraints that the software developer must accept as given (e.g., hardware, operating system) while others (e.g., algorithms, data structures) provide opportunities for the software developer to experiment with. The software developer must be aware of those factors that he can control and those that he cannot. The primary objective of this paper is to identify and list some (by no means all) of the main factors that affect the response time of a system and discuss some approaches that a software developer can take to meet the response time requirements of a realtime software.

4.0 APPROACHES TO HANDLE RESPONSE TIME REQUIREMENTS

For the purpose of illustration, let's first create a scenario and then discuss some approaches that a software developer might use. Suppose, your organization, a software house, has just finished developing a realtime software for a large financial institution that has several branches in the country. Overall, your client is satisfied with the quality of your company's software except for the response time. Your client has requested your organization to lower the maximum response time from the present 15 seconds to 10 seconds. The software project manager has asked you to handle this problem. What can you, as a software developer, do to solve this problem?

As mentioned earlier, numerous (often interacting) factors affect the response time of a system. Some of the factors are hardware, software, tools, methods, network architecture and topology, and system environment. We will discuss some aspects of these factors in the next subsections.

4.1 Hardware

The type of computer hardware used will invariably affect the response time of a system. Hardware characteristics include single versus parallel processors, RISC (reduced set instruction set computer) versus CISC (complex instruction set computer), processor speed, word length, RAM and cache size, bus (i.e., control, address and data) width and the speed of I/O devices. All these hardware characteristics will affect the performance of a computer system which, in turn, will affect the system's response time. For example, a multiprocessor will generally execute faster than a single or uniprocessor as it can perform many computations concurrently (i.e., in parallel). Similarly, if the type of data processed is largely simple (e.g., character or integer type). A RISC will execute faster as it requires fewer memory accesses. Likewise, a larger and faster RAM, cache, bus width, and I/O devices can improve the response time of a system considerably [11,13].

In a network environment, the hardware will also include the transmission medium and its capacity and speed, modems, multiplexers, bridges, routers and switches. The capacity and speed of these devices will undoubtedly affect the response time of a system [10,14,15].

A software that fails to meet the timing requirements with one type of hardware configuration may satisfy the requirements if the hardware is replaced with a different configuration. If the cost of changing the hardware configuration is cheaper than the cost of modifying a complex piece of realtime software that is correct and acceptable in every respect except the response time requirement, it may pay the software developer (or the client) to simply replace or upgrade the hardware configuration. The realtime software can go into production straightaway thus enabling the client to perform his business operations sooner. If this is a novel application, the client may gain an edge over his competitor.

However, replacing or upgrading computer and communications hardware for the sake of one realtime application software may not always be feasible as it can be quite costly (e.g., replacing an expensive mainframe computer or communication devices). It can also be costly from the point of view of time and inconvenience. Changing the hardware configuration may also affect the existing applications.

4.2 Operating System

A realtime software, like any other software, must run under an operating system platform. If the operating system is slow, the realtime software that runs under it will also be slow and vice versa. Therefore, an operating system that runs time-critical applications must be efficient in order to satisfy the response time requirements.

An operating system consists of several modules such as process, memory, I/O and file management modules. All these modules must be designed efficiently in order to provide the requirements of a realtime software.

Some of the main operating system components that affect system performance are: process switching, interrupt processing, memory management, file management and input/output. A multitasking or multiuser operating system must manage several processes concurrently, i.e., it must interleave the execution of several processes. This requires switching processes, e.g., when a process is blocked for I/O, the processor can work on another process. Process switching, like any other operating system function, is actually an overhead, but a necessary one. To switch to another process (in the ready queue), the operating system must save the state (e.g., process control block, stack) of the current process so that it can be safely restarted at a later time. It also requires the restoration of the state of the switched process. All this takes up processor time. As an operating system must switch processes frequently to provide better throughput and to optimize the use of system resources (such as processor, memory, and I/O devices), its implementation must be very efficient. The thread concept used in modern operating systems (such as OS/2, Windows 95 and Windows NT) promises to reduce this switching overhead to some extent. Related to process switching is the context switching. This also requires a time overhead, but it is less compared to process switching [11,13].

Another area of concern is interrupt processing (including multiple or nested interrupts). Like process (or context) switching, interrupt processing is also a necessary overhead. As interrupts occur very frequently, they must be processed efficiently. This can be achieved by coding the interrupt handlers efficiently. Also, to meet the response time requirements, interrupts associated with realtime applications must be given higher priorities [7,11,13].

I/O techniques, e.g., programmed I/O and direct memory access (DMA) can also influence the speed of processing. For example, if the amount of information read from or written to a disk is large, the DMA technique would be a better choice than the programmed I/O.

Buffering can also affect the response time. All I/O requires some kind of buffering. A system can use one, two or multiple buffers. Using many buffers will obviously improve the processing and hence the response time. The method used in scheduling disk access (e.g., LRU, SCAN, C-SCAN) will also affect the response time. Thus it is important to choose one that gives the best performance [11,13].

Similarly, the memory management technique used can affect the response time. For example, in virtual memory management, the page size, number of frames allocated to a process and page replacement algorithm will all have a bearing on the response time. Some may give better response time than others.

The file access method (e.g., sequential, indexed, direct) and the file allocation method (e.g., contiguous, linked or indexed) used can also affect the response time. Which file access method is better depends, to some extent, on the application. If an application requires only retrieval, the direct access method (or hashing) may be a better choice. However, if it requires, reading and writing records, the indexed file allocation method may be better than the others. Similarly, for sequential access, the contiguous method may be a better choice [11,13]. It is not always easy to determine which one is the best as all these access methods may be used in a computer environment.

Operating systems are designed for all types of users. Therefore they have to be quite general. A general operating system will meet the needs of most organizations. However, it is likely to be a little inefficient for any particular organization as the organization may not require all the features contained in the operating system. It may just require some features, but implemented very efficiently. Thus, if an organization requires fast response times, it might want to consider fine-tuning the existing operating system or perhaps develop its own customized realtime operating system (RTOS).

As you can see, the software developer may modify the operating system in one or more ways to achieve his purpose, i.e., to meet the response time requirement. Or, he may not have much choice in the matter if the client chooses not to alter the operating system (maybe because it may affect other existing applications). Also, changing or fine-tuning an operating system (especially a mainframe operating system) can be an expensive exercise.

4.3 Languages

The language used to develop a realtime application may also affect the response time. Programs written in assembly language will generally execute faster than those written in higher level languages such as 4GLs. Similarly, a strongly typed language such as Pascal will execute faster than a weakly-typed language such as C++, as the former does not require type-checking [6,7,8].

Some compilers will optimize and generate efficient code and thus improve the speed of execution. They may also generate code that makes use of a machine’s internal registers which can further improve the execution speed. The two C++ programs given in Fig. 1 illustrate these concepts.

Program 1 is inefficient compared to Program 2. In Program 1, the statement **num=5;** is executed 1000 times as it appears inside the for-loop. An optimizing compiler would remove the statement and place it before the for-loop as shown in Program 2. Also, Program 1 does not use any register variables. Declaring **i** and **num** as register variables will improve the processing speed.

If the computer has multiprocessing capabilities, a concurrent language (e.g., Concurrent C++ or Concurrent Pascal) can help to reduce the response time. This is illustrated in Fig. 2. The syntax **cobegin-coend** (i.e., concurrent begin - concurrent end) in Program 2 is not part of C++. It is used here for the purpose of illustration only.

In Program 1, the first four assignment statements are executed sequentially whereas in Program 2, they are executed concurrently (assuming there is a multi-processor). Thus, using a concurrent language the response time can be improved considerably especially if the tasks can be performed concurrently (as in processing large arrays or simulation modelling).

```
// Progam 1 - inefficient
#include <iostream.h>
main()
{
   int i, num;
   longint sum=0;
   for (i=0;  i<1000; i++)
   {
      num  =  5;     //  assigns  inside loop
      sum = sum + i*i  + num;
   }
   cout << "\n" << sum;
   return 0;
}
```

```
// Program 2 - efficient
#include <iostream.h>
main()
{
   reg int i, num;  // uses registers
   longint sum=0;
   num = 5;         // assigns outside loop
   for (i=0;  i<1000; i++)
      sum = sum + i*i  + num;
   cout << "\n" << sum;
   return 0;
}
```

Fig. 1: An efficient and an inefficient program

```
// Program 1 - sequential
#include <iostream.h>
main()
{
   int a, b, c, d, sum;
   a = 1;
   b = 2;
   c = 3;
   d = 4;
   sum = a+b+c+d;
   cout << "\n" << sum;
   return 0;
}
```

```
// Program  2- concurrent
#include <iostream.h>
main()
{
   int a, b, c, d, sum;
   // All statements in the cobegin - coend
   // block are executed concurrently
   cobegin
      a = 1;
      b = 2;
      c = 3;
      d = 4;
   coend;
   sum = a+b+c+d;
   cout << "\n" << sum;
   return 0;
}
```

Fig. 2: A sequential and a concurrent program

4.4 Databases

A network database can be centralized or distributed. A centralized database stores all the records in one location (or node) whereas a distributed (or decentralized) database stores the copies of the same database in different locations. Each has its advantages and disadvantages. A centralized database is more suitable if updating occurs frequently as it requires changing in only one location. However, accessing it from a remote location will take longer time (and will also cost more as it requires transmitting data over a network). A distributed database in this case will reduce the remote access time (and cost). However, updating it can be quite time-consuming as changes must be made simultaneously in all locations. All accesses to the database copies must be denied until they are all updated and verified. This requires quite a bit of coordination between the computers in various locations. It is usually implemented by maintaining a master clock and using some kind of time stamping. As the implementation itself requires transmission of control data, it is unlikely to be efficient from the point of view of response time requirement [5,7].

Another database approach that can be used is the partitioned or segmented database. A segmented database is stored in different places. Only a segment, i.e., part of the database, is stored in any one location. Typically, the records that will be accessed most frequently in a particular geographical location are stored in that location. This approach will generally improve the response time (besides reducing transmission cost). Thus this approach may be more appropriate for fulfilling the response time requirements [5].

The client or the software developer must therefore consider the above factors when designing databases for realtime applications. The client/server model addresses this issue to some extent. It attempts to distribute the processing functions optimally across a network. The workstations generally provide the graphical user interface required by end-users while the server provides the database management functions [13,14].

.

4.5 Algorithms

Algorithms are important in the design of any software, especially realtime software. The same task can be accomplished in one or more ways, i.e., by several algorithms. But some algorithms are more efficient than others. For example, there are many sorting algorithms that one can use, but some (e.g., Hoare's quicksort) are clearly more efficient than others. Similarly, there may be many to traverse a tree-type data structure, but some may be better than others for any given application. For example, the inference engine of an expert system can use the depth-first, breadth-first or best-fit approach to arrive at a solution. The expert system software developer must experiment and select an algorithm that best suits his needs.

There is also a processing overhead (e.g., stack management) associated in a function or procedure call [13]. Although modularisation is good from the software design point of view, it can nevertheless degrade the response time of a system, especially if it has too many functions or procedure calls (such as those applications that require the use of recursion).

Using macros instead of function or procedure calls can improve the execution (and hence the response time) as they do not incur the overhead of function or procedure calls. The macros are simply substituted in the code wherever they appear or are referenced in the source program. Using macros, however, will increase the amount of memory required to store the program, especially if they are called or referenced in too many places in the program. Similarly, the use of inline functions in object-based programs will achieve the same purpose. The software developer must consider these factors when designing a realtime software.

4.6 Data Structures

The type of data structures used can also affect the response time. For example, using a variable type of register or unsigned short integer for a loop index may be faster than just using an integer variable. Similarly; using a one-dimensional array will be faster than using a two-dimensional array; an array, faster than a linked list; and a numeric type data, faster than a character type data [6,7,8].

4.7 Network Factors

There are many network factors that can affect the response time of a system, e.g., type of channel (e.g., leased, switched) used and its speed, attenuation, transmission techniques (e.g., multiplexing, com-pression, circuit/packet switching), type of modems, bridges and routers and their speed, number of nodes in the network, network architecture (e.g., LAN, MAN, WAN) and topology (e.g., bus, star, ring), transmission techniques (e.g., frame relay, asynchronous transmission mode), and network standards (e.g., OSI, ISDN) and compliance to these standards [14,15]. It is beyond the scope of this paper to discuss all these factors that will affect, to one degree or another, the response time in a distributed system. Here, we will merely discuss a few.

The type and speed of transmission channel used will certainly affect the response time. Using a leased line (even it costs a little more) will undoubtedly improve the response time compared to using a dialup or switched line as the latter requires call setup time. The bandwidth of a channel will also influence the response time - the larger the bandwidth, the better will be the response. You will also need faster modems to go with the high bandwidth. The type of transmission medium used will also affect the response time. Currently, fiber optics offers the highest speed compared to other media such as twisted-pair or coaxial cable. Additionally, instead of using the analog telephone line (which requires signal conversion), an organization may decide to go all digital, e.g., by using fiber optics. This will not only result in faster transmission but will also reduce the error rate. The combined effect of increasing the data rate and reducing retransmission (as a result of lower error rate) will improve the response time considerably in distributed processing [9,10,14,15].

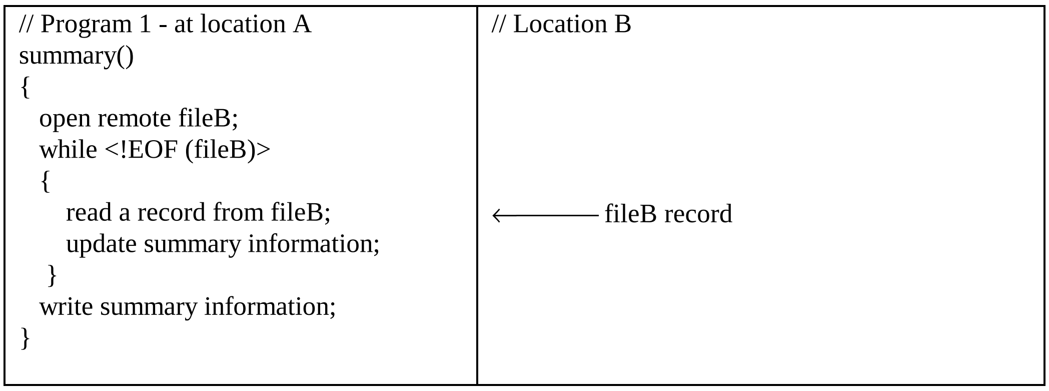
Data compression and multiplexing techniques, even if they incur some overhead, will also help to improve the overall data rate in a distributed system, which in turn, will reduce the response time.

Using standards such as OSI and ISDN will likewise improve the data rate (and hence the response time) as they will minimize the amount of conversion work needed to transmit messages on a network. Using intelligent routers will also improve the response time as they will route data using the best possible paths, i.e., paths which are least congested.

The level of data encryption/decryption for transmission over a network can also affect the response time. That means, encryption must be kept to the minimum (although this may cause security problems in some applications). Similarly, too much error checking can also reduce the transmission rate and hence the response time. The newer transmission technologies such as frame relay and cell relay (asynchronous transmission mode) minimize such error checking by simply trusting in the reliability of the transmission media (whose quality has certainly improved over the years). Thus by providing better data rates, these newer technologies can help to minimize network delays [14.15].

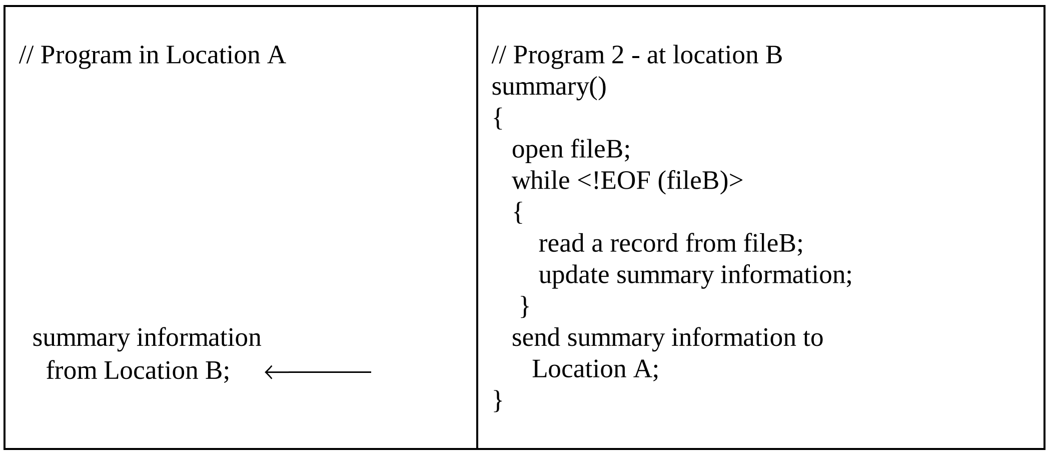
A distributed realtime application may also use modern techniques such as remote procedure call (RPC) or process migration (PM) to improve the response time [1,4,9,13,14]. To illustrate, let's suppose a user submits a query to the system in Location A to obtain summary information of a database stored in Location B (a remote location). This can be accomplished either by using either Program 1 (Fig. 2a) or Program 2 (Fig. 2b). If the database is large, the second approach is clearly faster (and cheaper as it reduces the communication cost).

Process migration is quite similar to remote procedure call except that the process itself is migrated from Location A to Location B and executed there. One reason why a process may be migrated from one location to another is to achieve network load balancing. Perhaps the current workload at Location A is very heavy and this may cause a process to actually take a long time to respond to a user request. In this case, it might be expedient to just migrate that process to Location B so that it can execute them faster [1,2,10,13,14]. These techniques - remote procedure calls and process migration - will become more popular as nations move towards information superways using advanced technologies such as optical fibers, ATM and ISDN.



```
// Program 1 - at location A              // Location B
summary()
{
   open remote fileB;
   while <!EOF (fileB)>
   {
      read a record from fileB;           <———— fileB record
      update summary information;
    }
   write summary information;
}
```

Fig. 2a: Algorithm without RPC



```
// Program in Location A                  // Program 2 - at location B
                                          summary()
                                          {
                                             open fileB;
                                             while <!EOF (fileB)>
                                             {
                                                read a record from fileB;
                                                update summary information;
                                              }
  summary information                        send summary information to
    from Location B;    <————                   Location A;
                                          }
```

Fig. 2b: Algorithm with RPC

5.0 FACTOR INTERACTIONS

So far, the paper merely presented possible 'stand-alone' solutions to the response time requirement. Clearly, the above solutions can be used in concert (i.e., combined) to achieve the desired minimum response time. For example, multiplexing and compression techniques can be used together to achieve a higher data rate. Similarly, efficient algorithms and data structures can result in a better response time. A combination of all the above solutions can further improve the response time. However, to experiment with all these factors is not a trivial task. There are too many combinations to consider. It is a combinatorial problem requiring much study and analysis. The software developer must use techniques such as queuing and simulation to study and analyze the response times. In this way, it is possible (even though costly) to satisfy the response time requirements in distributed realtime applications.

6.0 CONCLUSION

As mentioned at the beginning of the paper, there is a growing demand for distributed realtime applications. Such applications require fast response times as users or machines must interact with computers to obtain up-to-date information. A realtime software is considered defective, and therefore unacceptable, if it passes all the requirements (e.g., logic, accuracy, human interface) but fails to meet the response time requirements of the client. In other words, a realtime software must be very responsive to the timing needs of the application. That means, software developers must make special effort to incorporate techniques that will meet the timing requirements of the client. This paper has identified a list of factors (not comprehensive by any means) that a software developer must consider. Then it went on to discuss some possible approaches and techniques that the software developer can use to meet the inherent timing requirements of a realtime software.

REFERENCES

[1] Y. Artsy, and R. Finkel, Designing a Process Migration Facility, *IEEE Computer*, Vol. 22, 1989.

[2] A. Burns, and A. Wellings, Real-Time Systems and Their Programming Languages, Addison-Wesley, 1990.

[3] Ian Sommerville, Software Engineering. Addison-Wesley, 1993.

[4] C. L. Liu, and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of the ACM,* Vol. 20, 1973.

[5] I. Martin, Analysis And Design Of Business Information Systems. Prentice-Hall, 1995.

[6] Pfleeger, Software Engineering: The Production of Quality Software. Maxwell Macmillan International, 1991.

[7] R. S. Pressman, Software Engineering: A Practitioner's Approach. McGraw-Hill Inc., 1992.

[8] Schach, Software Engineering. IRWIN, 1993.

[9] K. Schwan, and H. Zhou, Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads, *IEEE Computer,* Vol. 18, 1992.

[10] K. Shin, HARTS: A Distributed Real-Time Architecture, *IEEE Computer*, Vol. 24, 1991.

[11] A. Silberchatz, and P. Galvin, Operating Systems Concepts. Addison-Wesley, 1994.

[12] A. S. Tanenbaum, Distributed Operating Systems. Prentice-Hall, 1995.

[13] William Stallings, Business Data Communi-cations. Prentice-Hall, 1995.

[14] William Stallings, Operating Systems. Prentice-Hall, 1995.

[15] William Stallings, Data And Computer Communications. Prentice-Hall, 1994.

BIOGRAPHY

Palaniappan Sellappan obtained his BEc from the University of Malaya in 1970, his MSc in Computer Science from the University of London in 1973 and his PhD in Information Science from the University of Pittsburgh in 1978. Currently, he is an Associate Professor in the Faculty of Economics & Administration, University of Malaya. His research areas include object-oriented technology, decision support systems, business data communications, EDP Auditing, database, software engineering and simulation. He is a member of the British Computer Society and The Engineering Council (UK).