A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation,
Quality and Testing, Robotics
May 23 – 25, 2002, Cluj-Napoca, Romania

REAL-TIME DISTRIBUTED MULTIMEDIA PROCESSING

Authors: Gheorghe Sebestyen Lecturer, Cristian Trebea student

Technical University of Cluj-Napoca, Computers Department
E-mail: Gheorghe.Sebestyen@cs.utcluj.ro

ABSTRACT

Modern Internet communication often involves timely multimedia data transmission, processing and visualization. Real-time multimedia processing requires high performance capabilities that, usually, exceed today's desktop computers' possibilities. This article presents a distributed multimedia processing solution that enhance the performance of a single desktop computer by using the idle processing capabilities of other computers connected on the same local network. The basic idea is to dynamically distribute multimedia processing tasks and data to a number of partner computers and execute them in a parallel fashion. A special scripting language was developed to describe the processing sequence in a way that allows distributed execution.

Keywords: multimedia, real-time, distributed execution

1. INTRODUCTION

Today, distributed processing is one of the key-solutions for complex, regionally spread applications, from usual Internet communication services, to databases, e-commerce or e-business. Distribution offers a number of advantages such as: a higher reliability and availability, a better use of existing resources, remote access to common data, higher performance at low cost and many others. But distributed design requires new application models, methodologies and tools. New complex tasks, like process synchronization, concurrent access to common data, data consistence, communication delays and error masking, etc., must be solved in a transparent way.

In the last ten years a number of new technologies were developed, as support for distributed application design. Some of the most used technologies are: client/server-base services, RPC (Remote Process Control), COM, DCOM, and CORBA and web-protocols (HTTP, CGI, XML, etc.). These technologies offer transparent access to remote data, processes, or services, which means that the end-users are not aware of the distributed nature of the requested services.

There are still few solutions for real-time distributed applications. Communication delays are difficult to mask in time-critical applications, such as multimedia communications or remote process control. In multimedia applications a constant and highly periodic data flow must be assured; in remote control closed loop deadlines must be guaranteed. The design of such applications is more difficult if complex data processing tasks must be executed in predefined time limits.

This article presents a distributed multimedia-processing model that increase the responsiveness of a desktop-station involved in a time-critical multimedia

communication. The performance of the desktop-station is enhanced by distributing the multimedia processing tasks to partner desktop-stations.

2. PROBLEM DEFINITION

The goal of the present research work is to develop a framework for distributed processing that improves the execution time of relatively complex multimedia processing task sequences.

Usually, multimedia processing implies quasi-standard procedures such as: periodic data reception and transmission, filtering, data coding and decoding, compression and decompression, visualization, etc. These procedures are applied to an input data stream continuously, in a predefined sequence. At the end, an output data stream is generated. The processing sequence must be executed in the shortest time. The time will depend on the complexity of the processing sequence, the quantity of input data and on the available processing resources. A "best-effort" solution is requested under the above conditions.

The system must have an input data channel, an output data channel, which may be configured as communication channels (network sockets), as files, or as input/output interfaces (e.g. web-camera, video terminal). The data processing sequence is predefined in a scripting language. The language must offer support for periodic and continuous data processing. The data must be structured in frames specific for multimedia data (video and audio data).

The system must have the possibility to distribute and collect data to and from computers organized in a processing cluster. It also must have the means needed to distribute processing tasks, to activate and stop them remotely and to synchronize and reorder the results. Task allocation must depend on the available resources and on effective workload of the partner computers. The allocation scheme should offer the best response time for a given resource configuration.

3. THE FRAMEWORK'S ARCHITECTURE

Figure 1 presents a generic scheme of the proposed framework. The system is composed of a Coordinator computer and a number of Worker computers. The coordinator is responsible for receiving the input data stream, for generating the output data stream and for allocating processing tasks to Worker computers. A Worker is a computer connected on the same local network with the Coordinator, and is "willing" to execute processing tasks.

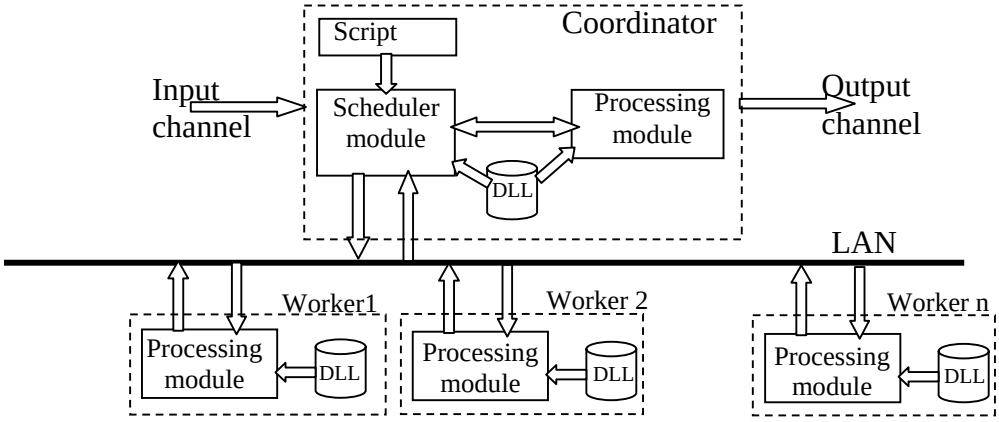

Figure 1 Bloc scheme of the proposed framework

The Coordinator contains a Scheduler module, a Processing module and a Library of multimedia processing tasks (DLL). The Scheduler is the main part of the framework, that solve the following tasks:

- parses the Script that describes the input and output data formats and the sequence of operations needed to process the input data
- analyzes data dependences and determine possible micro-sequences that may be executed in parallel
- determines the processing cluster's configuration (number of Worker computers and their workload)
- allocates data frames and processing tasks to Workers based on an optimization policy
- gathers the results, reorder them in accordance with the original Script and generate the output data stream

The Library contains a set of predefined multimedia procedures organized as a dynamic library. These procedures are the building blocks of a processing sequence; they are called from the Script. The Coordinator contains a complete library with all the predefined procedures. During execution, some of the procedures are loaded on the workers' local libraries if the scheduler allocates the corresponding operations to those computers.

A Worker computer contains a Processing module and a Library of local procedures. The Processing module receives requests from the Scheduler to process data frames using some predefined procedures. If a requested procedure is not contained in the local library than a procedure-download sequence is initiated. The procedure is transferred from the Coordinator to the Worker. If later the procedure is required again, there is no need for a new download.

A Processing module may be contained even in the Coordinator computer. For some small processing sequences it is not efficient to transfer a significant quantity of data to another computer, so it is executed locally. The Processing modules are "sleeping" memory resident tasks, activated whenever a new request is received from the Scheduler. They may be executed as background tasks when the computer is idle.

The scheduler uses a periodical pooling procedure to detect the active Processing modules. These modules form a processing cluster. The cluster's configuration may be changed during the execution of a script. The scheduler reorganizes the workload whenever a module becomes unavailable.

4. THE PROCESSING SEQUENCE DESCRIPTION LANGUAGE

A special language was developed to describe the data structures and processing sequences involved in a multimedia application. This language contains specific elements for multimedia data transmission, manipulation and processing. It also contains instructions for system initialization, configuration management and task distribution,

The language accepts four types of variables:

- local variables - used as temporary data storage inside of a single loop step
- global variables – used to transfer data between different steps of the execution loop
- file IO variables – used to denote an input or output file
- network IO variable – used to denote a communication channel

A Script describing a specific processing sequence has two parts: an initialization part and an execution part. The initialization part is used to set the system's configuration flags (e.g. timing limits, synchronous/asynchronous data access, task allocation policy, etc.), to declare variables and to specify the input and output data streams.

The execution part consists of a main execution loop that contains executable instructions. The following instructions are defined:

- get(stream_var, memory_var, size) – transfers a block of data of the specified size from the stream_var (file IO or network IO) to the memory variable (local or global)
- skip(stream_var, size) – moves the stream pointer forward with a number of locations equivalent to the size of a data block
- exec (destination_memory_var, source_memory_var, dll_procedure) – applies the dll_procedure to the source variable and the result is placed in the destination variable; the variables are data blocks
- put(stream_var, memory_var, size) – transfers a block of data of the specified size from the memory variable into the stream variable

The "exec" instruction is the only one that may be allocated to different worker computers. The scheduler module distributes data and activates dll procedures in remote locations. Based on the data flow analyses, data dependences are detected and parallel executions are lunched. More steps of the main loop are executed in parallel. The results may be generated in an "out of order" manner. The scheduler reorders the data in the output stream in accordance with the original order specified in the sequential script.

5. TASK ALLOCATION POLICY

An important factor in the efficiency of the proposed distributed processing framework is the task allocation policy. The scheduler must generate a task allocation scheme in accordance with the available resources and with the required processing power. The communication delays and the data transfer times also mast play an important role in the final decision. A processing cluster may contain computers with different resources and performance parameters. Also the number of computers in the cluster may change dynamically. Different multimedia processing tasks require different execution times. The execution time depends on the computers' performance characteristics.

In the present implementations two allocation policies are used: a simple uniform load algorithm and a time-based algorithm. The first algorithm is recommended when the computers of the processing cluster have similar performance characteristics. The idea is to distribute tasks equally between the computers of the cluster. Some data dependences are taken into account to avoid unnecessary data transfers; tasks working on the same dataflow are allocated to the same computer.

The time-based algorithm is more complex and offers a better efficiency when the computers of the cluster have different characteristics. The scheduler measures the response time of every computer for a standard task and measures the transmission time of a predefined block of data through the network. Also the nominal execution times of different multimedia procedures are taken into account. Based on this information the scheduler generates an optimal task allocation scheme. The allocation algorithm computes the delays for different allocation solutions and selects the one with the shortest delay. It is a step-by-step algorithm, in which the worker computers are

progressively loaded. For simple procedures it is not efficient to transfer a significant amount of data through the network. It is also possible to execute such a procedure by the local processing module.

6. EXPERIMENTAL RESULTS

The implemented multimedia-processing framework was tested under a number of conditions. The following parameters were modified:

- the complexity of the processing sequence,
- the size of a data block
- the transmission frequency of the network
- computers with different performance characteristics

The measurements were compared with a single computer version. In most cases the execution time was smaller. The time reduction was more significant for complex processing sequences and when high seed networks were used. For simple processing sequences the scheduler decided to execute the operations locally, avoiding the transmission delays.

A significant improvement was experienced in video application, where a sequence of image filtering and enhancement operators was applied to a stream of video frames. For the best configurations a 2 to 3 times execution time reduction was obtained. This enhancement in processing speed allowed to implement on-line applications, which otherwise, on a single standard desktop computer wouldn't be feasible.

7. CONCLUSIONS

This article presents a network-based, distributed framework that improves the responsiveness of a multimedia processing system. The basic idea is to distribute multimedia-processing tasks' execution into a cluster of computers connected on a local area network; a significant execution time reduction is obtained through parallel execution of tasks. A special scripting language was developed and used to describe the processing sequence. Through a parsing and data flow analyses the processing sequence is portioned in procedures that may be executed in parallel on remote computers.

A scheduling module is responsible for distributing tasks to the computers of the processing cluster. A uniform load or a time-based algorithm may be used as the allocation policy. The second one determines an optimal allocation solution for a given configuration and workload.

The experiments showed the feasibility of the proposed solution. Significant time reductions were measured for video stream processing. The presented distributed framework allows the implementation of some on-line processing sequences that cannot be implemented on single computer architecture.

The proposed solution may be extended to other applications where execution time is a critical factor.

8. REFERENCES

1. Coulouris G, Dollimore J, Kindberg T, [1994], Distributed Systems: Concepts and Design, *Addison Wesley*,
2. Hansson H, Gunningberg P, Flack H, [1994], Distributed Real-Time Systems *Technical Report, Royal Institute of Technology Stockholm*, Sweden

3. Kalogeras A, Stamatis K.L, Efstathiou K, [1993], An Implementation of an Intelligent Distributed Real-Time System, *Proc.of the 8th Int. Symposium on Intelligent Control*,Chicago, USA,
4. Shatz, Sol M., [1993], Development of Distributed Software: Concepts and Tools, *Maximilian Publishing Company*,
5. Ghe. Sebestyen , K. Pusztai, [1999], A Service-Based Distributed Real-Time *Control System,12-th International Conference on Control Systems and Computer Science, CSCS99*, Bucharest, pp227-231
6. Ghe. Sebestyen, G. Buzas, [1999], Distributed Services in Control Systems, *OSPMA-FieldComms99 Conference, Open Solutions for Process and Manufacturing Automation*, Telford, UK,
7. Stankovic, J.A, [1992], Distributed Real-Time Computing: Next Generation, *Jornal of the Society of Instrument and Control Engineers of Japan*,