Published in Proceedings of the conference
"Digital Compression Technologies and Systems for Video Communications" 4118, 146-153, 2000
which should be used for any reference to this work

Parallel Computation of Radio Listening Rates

M. Mazzariol, B. Gennart, R.D. Hersch
Ecole Polytechnique Fédérale de Lausanne, Switzerland[1]
Manuel Gomez, Peter Balsiger
Inst. de Microtechnique, Univ. de Neuchâtel, Switzerland
Markus Leder, Daniel Wüthrich
IBW AG, Turgi, Switzerland
Juerg Feitknecht
Liechti AG, Kriegstetten, Switzerland

ABSTRACT

Obtaining the listening rates of radio stations in function of time is an important instrument for determining the impact of publicity. Since many radio stations are financed by publicity, the exact determination of radio listening rates is vital to their existence and to their further development. Existing methods of determining radio listening rates are based on self completion diaries or telephonic interviews made with a sample population. These traditional methods however require the cooperation and compliance of the participants.

In order to significantly improve the determination of radio listening rates, special watches were created which incorporate a custom integrated circuit sampling the ambient sound during a few seconds every minute. Each watch accumulates these transformed sound samples during one full week. Watches are then sent to an evaluation center, where transformed sound samples are matched with the transformed sound samples recorded from candidate radio stations.

The present paper describes the processing steps necessary for computing the radio listening rates, and shows how this application was parallelized on a cluster of PCs using the CAP Computer-aided parallelization framework[2]. Since the application must run in a production environment, the paper describes also the support provided for graceful degradation in case of transient or permanent failure of one the system's components.

The parallel sound matching server offers a linear speedup up to a large number of processing nodes thanks to the fact that disk access operations across the network are done in pipeline with computations.

Keywords : parallel sound matching, parallel I/O, radio listening rates

1 INTRODUCTION

Obtaining the listening rates of radio stations in function of time is an important instrument for determining the impact of publicity. Since many radio stations are financed by publicity, the exact determination of radio listening rates is vital to their existence and to their further development. Existing methods of determining radio listening rates are based on self completion diaries or telephonic interviews made with a sample population. These traditional methods however require the cooperation and compliance of the participants.

In order to significantly improve the determination of radio listening rates, special watches[1] were created which incorporate a custom integrated circuit sampling the ambient sound during a few seconds every minute. The sound samples are split into sub-bands and converted into energy signals. Their envelopes are extracted, low-pass filtered and resampled at a much lower rate. Sound compression allows to store a full week of one minute records within the limited memory space of a watch. Watches are then sent to an evaluation center, where transformed sound samples are matched with the transformed sound samples recorded from candidate radio stations. Based on the matching performance, reliable listening rate statistics are established. Fig. 1 shows the complete data flow from the radio station emitter to the correlation center.

The evaluation center should be able to determine in *real time* the listening rates for a configuration of approximately 1000 watches and 100 radio stations. The computation of the listening rates for a single day (24h) should therefore last less then one day.

1. Contacts : Marc.Mazzariol@epfl.ch, RD.Hersch@epfl.ch

The present paper describes the processing steps necessary for computing the radio listening rates, and shows how this application was parallelized on a cluster of PCs using the CAP Computer-aided parallelization framework[2]. Since the application must run in a production environment, the paper also describes the support for graceful degradation in case of transient or permanent failure of one the system's components.

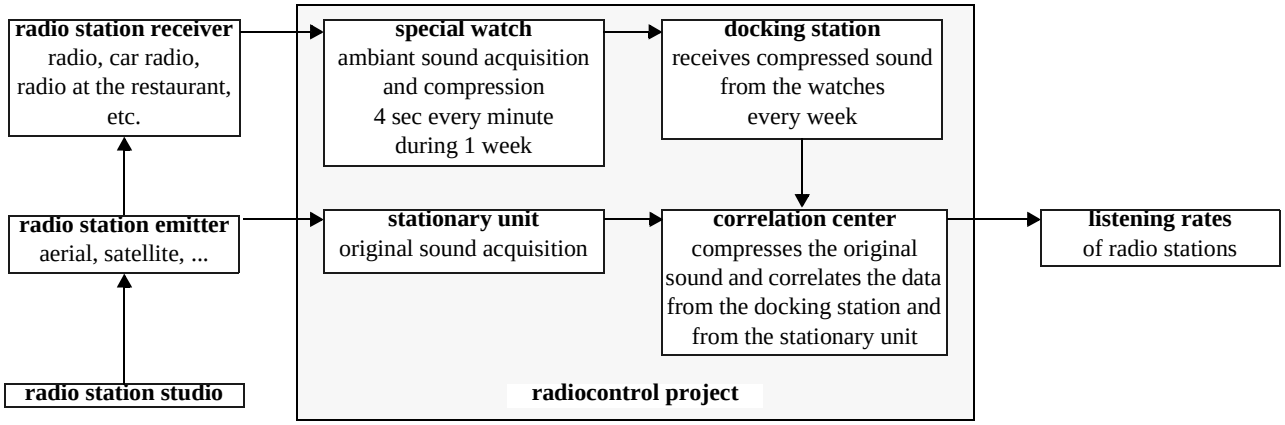

Fig. 1. Complete data flow from the radio emitter to the correlation center

2 THE MATCHING PROBLEM

In order to obtain the listening rates of radio stations for a given hour within a given day, one must match the transformed sound of the 1000 watches and of the 100 candidate radio stations. Each watch records the ambient sound during 4 seconds every minute. Each radio station is recorded during 10 seconds every minute. In both cases, sound is sampled at 3KHz. A small time shift may appear between sound acquisition in the watch and in the stationary unit, due to the uneven distance between them and the radio station emitter. In addition, the quartz inside the watch has a limited precision and the frequency may vary due to temperature variations. To reduce the time shift, the watches are synchronized every week (when they are sent back to the docking station) and a temperature sensor is incorporated in the watch to correct the clock deviation. In order to take into account the time shift, the discrete correlation between the compressed sound from the watch and from the stationary unit is done at successive 1 ms intervals, i.e. the matching is established by varying, millisecond per millisecond, the time position of the 4 seconds of sound acquired by each watch within a time window of 10 seconds. The largest match (correlation maximum) between watch and radio station is recorded for further evaluation, i.e. to determine for the considered minute and for a given watch whether the sound stems from one of the candidate radio stations. Figure 2 shows a global scheme of the matching procedure.

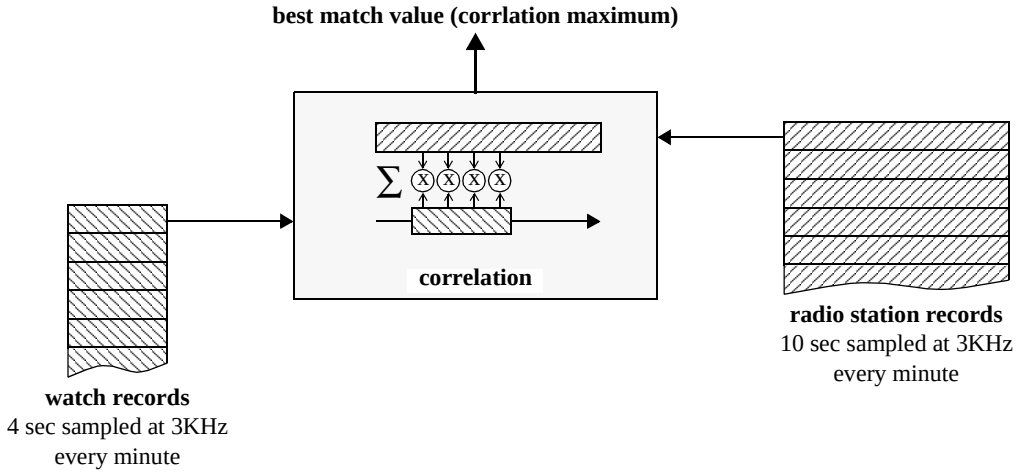

Fig. 2. Global scheme describing the matching process

2.1 Storage

The data required to perform the matching between ambient sound samples and radio station sound is organized in files. There is one radio station file per hour, which contains 60 minute records incorporating each 10 seconds uncompressed sound. Each minute record takes 60 KB. Therefore each hourly file has a size of 3.5 MB. Since we want to establish the listening rate for a full day (24 hours) and 100 radio station, we need 8.2 GB.

$$\text{one radio station minute record size} = 10 \text{ sec} \times 3 \text{ KHz} \times 2 \text{ Bytes} = 60 \text{ KB} \quad (1)$$

$$\text{one hourly radio station file size} = 60 \times 60 \cong 3.5 \text{ MB} \quad (2)$$

$$\text{total radio station file size} = 3.5 \times 24 \times 100 \cong 8.2 \text{ GB} \quad (3)$$

The watch file contains all minute records comprising 4 seconds of highly compressed sound for a full day, i.e. 24x60 records of 95 Bytes. Therefore each daily watch file takes 134 KB. Since we have 1000 watches, the total space required for the watch files is 134 MB.

$$\text{one watch minute record size} = 95 \text{ Bytes} \quad (4)$$

$$\text{one dayly watch file size} = 95 \times 24 \times 60 \cong 134 \text{ KB} \quad (5)$$

$$\text{total watch file size} = 134 \times 1000 \cong 134 \text{ MB} \quad (6)$$

Correlating one watch record with one radio station record produces a new result record of 8 Bytes (correlation maximum). Each result file contains all the result records for one hour, one radio station and all the 1000 watches. Therefore one result files has a size of 470 KB. Since there are 24x100 result files, the total required space for the result files is 1.1 GB.

$$\text{one result record size} = 8 \text{ Bytes} \quad (7)$$

$$\text{one hourly result files size} = 8 \times 60 \times 1000 \cong 470 \text{ KB} \quad (8)$$

$$\text{total result file size} = 470 \times 24 \times 100 \cong 1.1 \text{ GB} \quad (9)$$

By considering the total radio station file size (3), the total watch file size (6) and the total result file size (9), the total amount of data used to establish the listening rate for a full day (24 hours), 100 radio stations and 1000 watches is about 10 GB.

$$\text{otal size} = 8.2 + \frac{134}{1024} + 1.1 \cong 9.4 \text{ GB} \quad (10)$$

The radio station files, the watch files and the result file are all stored on a single PC file server. The correlation processes can freely access (read and write) this file server. Remote access to the file is achieved thanks to the NTFS distributed file system, which allows to share the physical hard drives of the file server over the local network. The full matching process for one day needs to transfer from the file server over the network a total amount of data of about 10 GB. The file server and the correlation process computer run under WindowsNT 4.0 and are connected through a Fast Ethernet network (100Kbits/sec) using the TCP/IP protocol.

2.2 Serial correlation algorithm

In serial processing mode, in order to obtain the result for one hour, 100 radio stations, and 1000 watches, the matching program incorporates the following steps :

1. Open and seek within each of the 1000 watch files to get the required target single hour records, i.e. 60 consecutive minute records of 95 Bytes (see (4)) from each watch file. This requires 1000 disk accesses, each one for reading $95 \times 60 \cong 5.5$ KB .
2. For all the 100 radio stations :
    2a. Load the hourly radio station file for the current hour. This step consists of a single disk access of 3.5 MB (see (2)).
    2b. Correlate the corresponding records from the hourly radio station file and the 1000 watch files. Since there are 60 records in the radio station file (one per minute) and each must be correlated with the corresponding minute record of each of the 1000 watches, i.e. 60x1000 = 60000 correlations need to be carried out. All the resulting match values (correlation maxima) are stored in local memory.

2c. Flush the resulting match values from the local memory to the remote result file on the file server. This step consists of a single write disk access of 470 KB (see (8)).

To establish the listening rate for a full day (24 hours), we need to repeat the steps listed above 24 times.

2.3 Serial performance analysis/measurements

The following serial performance measurements have been done with one Pentium-II PC 400 MHz file server and one Pentium-II PC 400 MHz correlation computer, both running under WindowsNT 4.0 and connected through a Fast Ethernet network (100 Mbits/sec). Let us analyse the time taken by each step described in the previous section. Step 1 requires 1000 disk accesses of 5.5 KB each. Since the data size is small and seek time is predominant, we obtain an effective throughput of 550 KB/sec, i.e. 10 sec are needed to complete step (1).

$$\text{total time for step 1} = \frac{1000 \times 5.5}{550} = 10 \text{ sec} \tag{11}$$

Step 2a is just a single disk access of 3.5 MB. At an effective throughput of 2 MB/sec, it takes about 1.75 sec. Since this step is repeated for all the radio station, we get the following total time :

$$\text{total time for step 2a} = 100 \times \frac{3.5}{2} = 175 \text{ sec} \tag{12}$$

The match time of two records is highly data dependent due to optimizations in the correlation procedure. Nevertheless, the mean time to correlate two records is 1.85 milliseconds. This value is correct for normal ambient sound. Step 2b requires 60000 correlations and therefore needs 111 sec to complete. Since this step is repeated 100 times, we get :

$$\text{total time for step 2b} = 100 \times 60000 \times 0.00185 = 11100 \text{ sec} \tag{13}$$

Step 2c is just a single disk write access of 470 KB. At an effective write throughput of 2 MB/sec it takes 0.25 sec. Since this step is repeated 100 times :

$$\text{total time for step 2c} = 100 \times \frac{470}{2 \times 1024} \cong 23 \text{ sec} = 2.664 \text{ hours} \tag{14}$$

The total time to establish the listening rate for 24 hours, 100 radio stations and 1000 watches is obtained by adding the previous results and multiplying them by 24 :

$$\text{total serial time} = 24 \times (10 + 175 + 11100 + 23) \cong 75.3 \text{ hours} \tag{15}$$

Since the I/O time represents less than 2% of the total time, the process is compute bound and no significant gain could be achieved by pipelining step 2a, step 2b and step 2c (i.e. making asynchronous accesses to the file server) in the serial implementation. Therefore the total processing time is approximately proportional to the number of watches multiplied by the number of radio stations, i.e. it can be deduced from equation (13).

$$\text{total serial time approximation} = 2.664 \times nbWatches \times nbRadioStations = 2.664 \times 1000 \times 100 = 74 \text{ hours} \tag{16}$$

The practical measurements are consistent with the previous analysis. The total computation time to produce the listening rate for one day exceeds the 24 hours limit. Therefore parallelization is needed.

3 PARALLELIZATION

3.1 The Computer-Aided Parallelization (CAP) framework

The CAP Computer-Aided Parallelization framework[2,3] is specially well suited for the parallelization of applications having significant I/O bandwidth requirements. Application programmers specify at a high level of abstraction the set of threads present in the application, the processing operations offered by these threads, and the flow of data and parameters between operations. Such a specification is precompiled into a C++ source program which can be compiled and run on a cluster of distributed memory PCs. The compiled application is executed in a completely asynchronous manner : each thread has a queue of input tokens[2] containing the operation execution requests and its parameters. Disk I/O operations are executed

2. A token according to the CAP terminology designates a serializable C++ data structure

asynchronously, i.e. while data is being transferred to or from disk, other operations can be executed concurrently by the corresponding processor. In a pipeline of disk access and process operations, CAP allows therefore to hide the time taken by the disk access operations if the process is compute bound. A configuration file specifies the mapping between CAP threads and operating system processes possibly located on different PCs. This configuration file is used by the run-time system to launch the parallel application. A Remote Shell Daemon (rshd) is located on each target computer to launch processes remotely and carrying out I/O redirection. The CAP framework works on several architecture (32 bits, 64 bits), several OS (WindowsNT, Unix, Linux) and uses TCP-IP sockets for communication.

3.2 Parallel correlation algorithm

The correlation application has been parallelized in a master-slave fashion (see Figure 3). The master only distributes jobs to the slaves, which are the effective workers. Since the master is idle most of the time, he resides on the file server computer. The *n* slaves are connected to the master through a Fast Ethernet network (100 Mbits/sec) using the TCP/IP protocols. All the computers are Pentium-II 400 MHz PCs running under WindowsNT 4.0. The radio station files, watch files and result files (total 10 GB) are stored in an array of Ultra Wide SCSI-II disks connected to the file server. The disks offer an effective throughput of 2 MB/sec each, the SCSI bus has a maximal throughput of 40 MB/sec and the Fast Ethernet network does not support more than 8 MB/sec.

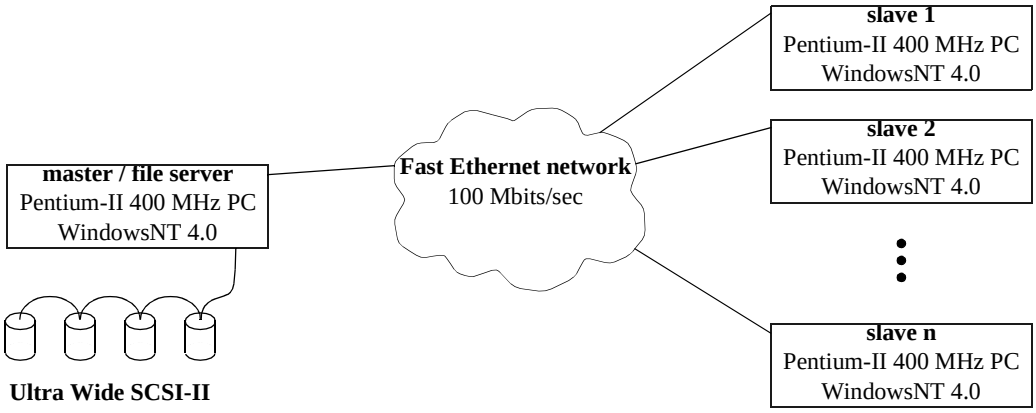

Fig. 3. Parallel architecture

The serial processing operations described in the previous section are parallelized by having different processors (*n* processors) working on different radio station files. The only change resides in step 2. Instead of making a simple loop over the 100 radio stations, each slave receives and computes a different iteration of the loop, i.e. works on a different hourly radio station file. Each slave reads a new radio station file and computes a new result file. This implies that the radio station and result files are read/written once during the whole program execution. When a slave starts to compute a new hour, he loads the corresponding watch minute records within the 1000 watch files. This correspond to step 1 and is done on every *n* slaves each time a new hour computation starts, i.e. each watch minute record is accessed *n* times, once by each slave. Therefore the watch files are loaded *n* times during the program execution. Equations (3) and (9) remain unchanged, but equation (6) must be multiplied by *n*. If *n* is small (near 10) the total amount of data transferred from the disks over the network will not increase more than 20%, which is acceptable. Equation (6) and (10) become respectively for $n = 10$:

$$\text{total watch file transfered size} = n \times 134 \text{ MB} = 10 \times 134 \cong 1.31 \text{ MB} \quad \text{(6bis)}$$

$$\text{total transfered size} = 8.2 + 1.31 + 1.1 \cong 10.6 \text{ GB} \quad \text{(10bis)}$$

Since there are 100 radio stations and 24 hours, the master distributes 2400 jobs to the slaves representing a total computation time of about 75.3 hours (see (15)). The CAP framework helps the programmer to distribute these jobs intelligently. When the computation starts, each slave receives a job from the master and starts computing it. Immediately after this first job distribution, i.e. while the slaves are still busy with their first job, the master sends a second job to all the slaves. This new job is not executed immediately but inserted in a job queue in each slave. As soon as a slave finishes computing his first job, he starts processing the second job (no idle time). During the execution of the second job, the master sends a third job to the slave and the whole process is repeated. This job distribution mechanism ensures that the slaves are never waiting for a new job, i.e. are always busy, and that the total computation time is load balanced between the slaves. Since each slave

receives a new job when he finishes computing the previous one, each slave works at its own speed; a faster slave will receive more jobs than a slower slave and since the jobs do not exactly require the same computation time, a slave can (for example) receive 2 large jobs while another receives 3 small jobs during the same time interval.

Since the radio station, watch and result files reside on a single computer, the file server treats the slaves read/write requests sequentially. In the previous section, equations (11), (12) and (14) shows that the total access time to the file server is $24 \times (10 + 175 + 23) \cong 1.3$ hour . This time is insignificant when compared with the 75.5 hours (see (15)) required to establish the listening rates in serial execution mode, but it may become significant in parallel execution mode, as stipulated by Amdahl's law. Therefore it is important to access asynchronously to the file server, i.e. read/write accesses should be hidden by the slaves computation time. The CAP framework helps the programmers in this task. CAP allows to assign the steps 2a, 2b and 2c to three different threads. This enables the execution of the three steps in a pipeline. While a job is requiring computation power and keeping the slave processor busy at step 2b, another job can make accesses to the file server (step 2a or 2c). In a totally transparent manner for the programmer, CAP ensures the correct data flow through the operations of the pipeline. Figure 4 shows the slave activity and resource occupation during the correlation process.

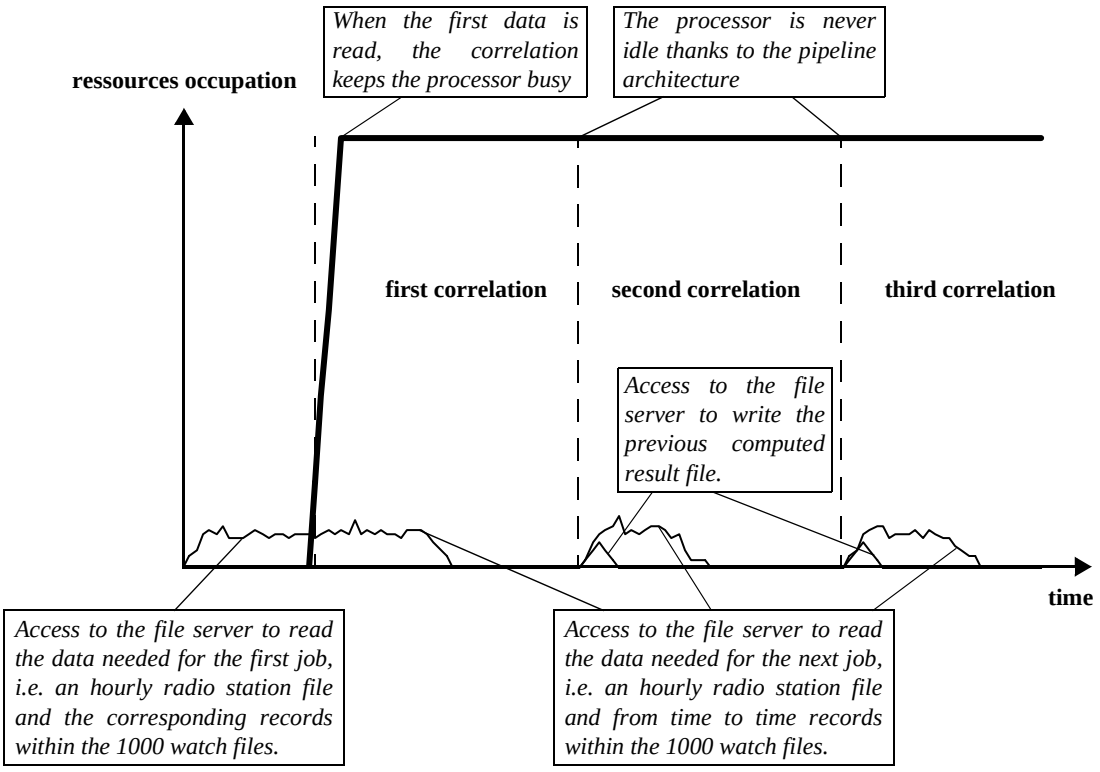

Fig. 4. Slave activity during the correlation process : the thick line represents the processor occupation on the slave, and the thin lines represent the file server accesses

3.3 Parallel performance analysis/measurements

Within a single processor, disk accesses are asychronous and executed in parallel with processing operations. After initialization of the pipeline, only the processing time, i.e. the correlation between radio station files and watch records determines the overall processing time. Therefore the time to perform the steps 1, 2a and 2c described by equations (11), (12) and (14) is hidden behind the processing time. Only the processing time of step 2b described by equation (13) remains. The total parallel time on *n* slaves and when doing asynchronous I/O accesses is given by :

$$\text{total parallel time} = \frac{24 \times 11100}{n} = \frac{74}{n} \text{ hours} \qquad (15\text{bis})$$

Figure 5 shows the computation time when increasing the number of slaves. The measurements have been done for 24 hours, 100 radio stations and 1000 watches on Pentium-II 400 MHz PCs. The performance results show a linear speedup,

i.e. processing time which is 74 hours on a single computer decreases to 24.7 hours with 3 PCs, 18.5 hours with 4 PCs (i.e. under 24 hours limit) and 14.8 hours with 5 PCs which is under the 16 hours time constraint imposed by the industrial specification. When the number of slaves becomes larger (above 20) the speed-up will not remain linear since the share file server will become the bottleneck.

Fig. 5. Speed-up diagram

4 GRACEFUL DEGRADATION IN CASE OF FAILURE

Since the presented application should work in a production environment, it is highly important to provide support for graceful degradation in case of transient of permanent failure of one of the system's components (failure of a slave PC for example). This support is based on the *checkpoint and restart* paradigm[4]. A watchdog process running in the master computer checks that at least once every 20 minutes, new results are written to the result file. If this is not the case, the radio listening rate computation process is killed on every compute node, the network is checked to verify which PC's are working, and a new configuration file is generated incorporating only those PCs which are currently living. The application is relaunched on the new configuration. The application first determines if the result file exists and which were the last consistent records written on file. The application then resumes the computation in order to generate the next matching records.

```
configuration {
pools :
        MasterLocation{ 123.456.78.00 };
        SlavePool       {123.456.78.01, 123.456.78.02};
processes :
        M       (MasterLocation,        Radiocontrol.exe);
        S1      (SlavePool,             Radiocontrol.exe);
        S2      (SlavePool,             Radiocontrol.exe);
threads :
        MasterThread                    (M);
        Slave[0].ComputeThread          (S1);
        Slave[0].IOThread[0]            (S1);
        Slave[0].IOThread[1]            (S1);
        Slave[1].ComputeThread          (S2);
        Slave[1].IOThread[0]            (S2);
        Slave[1].IOThread[1]            (S2);
};
```

Fig. 6. Typical configuration file for the radiocontrol parallel application.

Figure 6 shows a typical configuration file for the radiocontrol parallel application. This configuration file runs the application (Radiocontrol.exe) on one master operating system process (M) and two slave operating system processes (S1, S2). In the present example, we have one thread in the master (MasterThread) and three threads on each slave (on

S1 :Slave[0].ComputeThread, Slave[0].IOThread[0], Slave[0].IOThread[1]). The master is launched at the IP-address given by MasterLocation. The slaves are launched on any of IP-address in the SlavePool. If both computers from the SlavePool are living, S1 is launched on the first and S2 on the second. If one computer from the SlavePool is down then S1 and S2 are launched on the other living computer. This configuration allows the system to work properly in the case of one slave computing node failure. The master remains a critical resource (since it's the file server). Other configuration are possible.

5 CONCLUSIONS

In order to compute the listening rates of radio stations, we developed a parallel sound correlation program running on a cluster of PCs interconnected by Fast Ethernet. The parallel sound matching server offers a linear speedup up to a large number of PCs thanks to the fact that disk access operations across the network are done in parallel with computations. Support is provided for graceful degradation of the sound matching server. In the case of failure of slave computation PCs, the current computation is stopped, contributing processes are killed, the network is checked for living PCs and the application is automatically restarted on the new configuration.

The Swiss Radio company decided to adopt this system to measure the listening rates of radio stations in Switzerland[6]. The system listens to 120 radio stations and 30 televisions in 17 different studio locations. There are 800 test persons equipped with the special watch. The correlation is done with 9 slave PC's and 1 master PC, each one a Pentium-III 500 MHz computer. Commercial contacts with other countries have been established. A test panel has already been installed in Paris.

The project was a large collaborative effort with the contribution of many partners : the Institute of Microtechnics, Univ. of Neuchatel, for developing the custom integrated circuits for sound acquisition and compression in the watch[7], EPFL for parallelizing the sound matching algorithms[8], IBW AG for integrating all the software into the evaluation system[9], Liechti AG for creating the general concept and building the watches[10] and University of Zurich, Martin Bichsel, for creating the first version of the sound acquisition and compression software[5].

REFERENCES

1. S. Sun, C. Calame, M. Gomez, P. Balsiger, F. Pellandini, D. Wuthrich, P. Mishcler, J. Feitknecht, "Very Low-Power Terminal for Media Control Applications", International Workshop on Intelligent Communication Technologies, Cost Workshop 254, Neuchatel, Switzerland , May 5-7, 1999, see http://www-imt.unine.ch/cost/
2. Benoit Gennart, Roger D. Hersch, "Computer-Aided Synthesis of Parallel Image Processing Applications" Proc. Conf. Parallel and Distributed Methods for Image Processing III, SPIE International Symp. on Optical Science, Engineering and Instrumentation, Denver, Colorado, July 1999, SPIE Vol-3817, 48-61
3. V. Messerli, O. Figueiredo, B. Gennart, R.D. Hersch, "Parallelizing I/O intensive Image Access and Processing Applications", *IEEE Concurrency*, Vol. 7, No. 2, April-June 1999, 28-37
4. Jim Gray, Andreas Reuter, *Transaction Processing : Concepts and Techniques*, Morgan Kaufmann, 1992
5. Method for the compression of recordings of ambient noise, method for the detection of program elements therein, and device therefore, European Patent Application EP 0 887 958 A1, inventor ; Martin Bichsel, applicant : LIECHTI AG, issued 30.12.1998
6. http://tora.sri.ch/gd/fr/home/fr_textarchiv_1299.html#hörer
7. http://www-imt.unine.ch/Radiocontrol/
8. http://diwww.epfl.ch/w3lsp/research/gigaserver/
9. http://www.ibwag.com/radiocontrol/RC_MAIN.HTM
10. http://www.innofair.ch/technologiestandort/sp_d/99/pages/cebit_5.html