A tool for component-based schedulability analysis of distributed real-time pipelines

Nicola Serreli, Giuseppe Lipari, Enrico Bini
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {n.serreli,g.lipari,e.bini}@sssup.it

I. Introduction

In many scenarios, such as in the automotive context [1], [2], the complexity of developing a distributed real-time embedded (DRE) system is reduced by dividing the system into separate components, possibly developed by a different teams or third-party companies. The component-based approach does simplify both the analysis and the integration, although it introduce some waste of resource.

In distributed real-time embedded (DRE) systems, a software component is often modeled as a chain of tasks (also called *transaction* or *pipeline*) [3]. Each task of the pipeline is allocated on a (possibly different) processing node. The first task is activated periodically, or by external events characterized by a minimum interarrival time. The other tasks are activated in accordance with the chain order, i.e. they start upon the completion of the preceding one. The last task must complete within an *end-to-end* (EE) deadline relative to the activation of the first task.

In [4], [5], we presented a methodology to perform a component-based analysis of task pipelines. In our model, the interface of a pipeline component consists of a set of *demand bound functions* (`dbf`) [6], one for each node on which the pipeline runs. In [4] we presented a heuristic algorithm to assign intermediate task deadlines to minimize the `dbf` of the pipeline. In [5] we presented a methodology to compute the `dbf` of sporadic pipelines.

In this paper, we present a software tool for analyzing DREs. The tool consists mainly of a C++ library. The library models virtual processing nodes and task pipelines, and can perform both classical holistic schedulability analysis [7] as well as the algorithms described in [4], [5]. The library has been designed to be easily extended with new schedulability analysis, and more complex task models (e.g. DAGs). We show the internal structure of the library and show its usage on a simple example.

II. Structure of the software library

A. *Tasks, Transactions,* `dbf`

The library has been written in C++ and designed according to Object Oriented design methodology, to enhance modularity and extendability.

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7 under grant agreement n.214777 "IRMOS Interactive Realtime Multimedia Applications on Service Oriented Infrastructures" and n.248465 "S(o)OS Service-oriented Operating Systems."

The basic class `Task` models a sporadic/periodic non-concrete task (also called offset-free task). The class contains the main features of a task: worst-case execution time (`wcet`), relative deadline (`dline`), and period (`period`). To extend the `Task` class with other properties, it is possible to either derive a new specific class, or use the methods:

```
void set_property(string &s, double val);
double get_property(const string &s) const;
```

that associate the name of the property with a numerical value. For example, some algorithm could use the additional property `jitter`. If we want to specify that a task `tsk` has jitter equal to 3.5, we have just to invoke:

```
tsk.set_property("jitter", 3.5);
```

the algorithms can retrieve such information with:

```
j = tsk.get_property("jitter");
```

The `Task` class also exports some helper function to simplify the schedulability analysis algorithms implemented in the tool. These are:

- **`double`** `get_next_arrival(`**`double`** `t)`, which returns the next arrival time after `t`;
- **`double`** `get_num_next_deadline(`**`double`** `t)`, which returns the next absolute deadline after `t`;
- **`int`** `get_num_arrived_instances(`**`double`** `t1,` **`double`** `t2)` that returns the number of instances arrived in interval $[t_1, t_2]$, and
- **`int`** `get_contained_instances(`**`double`** `t1,` **`double`** `t2)` the ones entirely contained in $[t_1, t_2]$.

Finally, appropriate operators for reading/writing a task from/to a file are available:

```
ostream & operator<<(...);
istream & operator>>(...) throw(IllegalValue);
```

The class `Transaction` represents a chain of tasks. The way to create a transaction is to first create the `Transaction` object, and then repeatedly invoke the `add_task` method, specifying the index of the node on which the task has been allocated. Also, it is possible to read/write the transaction parameters from/to a file.

In this version of the tool, no allocation algorithm has been implemented yet. Therefore, we assume that tasks are already allocated to nodes by some other external tool.

The dbf of a task chain is represented by a function object called `dbf`. For example, it is possible to compute the dbf of a task by simply invoking the constructor:

```
dbf (const Task &t);
```

To obtain the value of the dbf in a certain interval of length t, it is sufficient to apply operator `()`. For example:

```
Task tsk(10, 20, 30);
dbf fun(tsk);
double v = fun(50);
```

will assign 20 to the variable `v`, since in the worst case there are 2 instances of task `tsk` in an interval of length 50.

We enable the combination of two dbfs by the operators:

```
dbf operator+(const dbf &d1, const dbf &d2);
dbf sup(const dbf &d1, const dbf &d2);
```

The first one returns the sum of the two dbf functions passed as parameters. The latter one returns the function that for every point t is the max of `d1(t)` and `d2(t)`.

Finally, the invocation of

```
bool check_sched(double alpha, double delta);
```

allows checking if the dbf is below the linear bound $\alpha(t - \Delta)_0$.

B. Assigning intermediate deadlines

The algorithms for assigning intermediate deadlines have been collected in the namespace `scan::opt`. We implemented a few versions of the simulated annealing algorithm which tries different alternative assignments until it finds a "good one" (not necessarily optimal). The versions only differ in the objective function to be minimized. We implemented the following ones:

- `optimize_slope()` minimizes $\max_{k,t}(\mathsf{dbf}(t)/t - U_k)$, where U_k is the total utilization of all tasks of the transaction on node k.
- `optimize_ratio()` minimizes $\max_{k,t}(\frac{\text{DBF}(t)/t}{U_k})$, i.e. we are minimizing the maximum ratio between the dbf in any point, and the value of the linear function of slope U_k.

Obviously, the two functions are only wrapper functions of a more general optimization that is not part of the interface. This general optimization function uses the GSL library[1] that already implements many optimization algorithms.

We also provided some sub-optimal polynomial function for assigning deadlines. The ORDER function [4] can be executed by

```
vector<double> ordered_assignment(...);
```

Unfortunately, with some pathological parameters the function may fail to find a solution. In such cases, the `NoSolution` exception is raised, and the user can revert to apply some simpler heuristic [8]:

[1]available as open source software at www.gnu.org/software/gsl/.

```
void slack_assignment(...);
```

or assign intermediate deadlines proportionally the tasks' WCETs by invoking:

```
void proportional_dlines(...);
```

C. Computing the sporadic dbf

The dbf of pipelines activated sporadically may exceed the one with strictly periodic activation [5]. We implemented the functions to compute the dbf in the sporadic case, that is

```
vector<dbf> spor_dbf_transaction(...);
```

The algorithm for computing the dbf of sporadic pipelines is rather time-consuming since it requires to span over all the possible arrival patterns [5]. Hence we implemented also the following two upper bounds to the exact dbf:

```
vector<dbf> test_ub_dbf_transaction(...);
vector<dbf> test_iub_dbf_transaction(...);
```

D. Holistic Analysis

In addition to the analysis based on the demand bound function, we have implemented also the holistic analysis described in [7]. This analysis is global, so it can only be executed on the whole set of transactions in a system. The holistic algorithms have been implemented in a separate namespace `scan::EDFHolistic`. The analysis is invoked by the function

```
bool offsetAnalysisTransMax(...);
```

which corresponds to algorithm MDO-TO in the paper. The function returns `true` is the set of transactions is schedulable, `false` otherwise. If all the pipelines are schedulable then the function `offsetAnalysisTransMax` stores the response times of the tasks in the `Transaction` classes.

Finally the function

```
vector<Transaction> plainTransactions(...);
```

can randomly generate set of pipelines. It reads the following input parameters: desired number of transactions (`numTrans`), minimum and maximum number of tasks per transaction (`numTasks_min`, `numTasks_max`), total utilization (`U`), minimum and maximum transaction period (`minPeriod`, `miaxPeriod`), the greatest common divisor among all transaction periods (`GCD`), the minimum and maximum end-to-end deadline (`minDead`, `maxDead`), and the number of processors (`numProc`).

III. Case study

To illustrate the tool developed, we present a simple case study (the source code is fully available at retis.sssup.it/~bini/sw/). This example can be run by executing `dbfs_try` after a proper configuration, with `./configure`, and compilation with `make`). We assume to have two pipelines distributed among a set of CPU. The number of tasks and the number of available CPUs are read from standard input. The output contains a considerable amount of data, such as the demand bound function on each node computed considering the initial deadline assignment and the ones computed with ORDER deadline assignment [4]. The data extraction is performed by php scripts. We choose php because it can be run from command line, it easy to read and can elaborate text with few lines of code.

All the steps, except the compilation, are collected together in one bash script `dbfs_tests.sh`, that runs the analysis, extract all the computed `dbf`s and invokes gnuplot to generate comparative diagrams, that show the difference among different methods used to assign the deadlines: initial (i.e. random) assignment, ORDER assignment [4] and slack assignment [8].

In Figure 1 we plot the linear upper bounds to the `dbf`s, as generated by gnuplot. We can observe that only the ORDER deadline assignment produces a schedulable transaction (with slope not larger than 1).

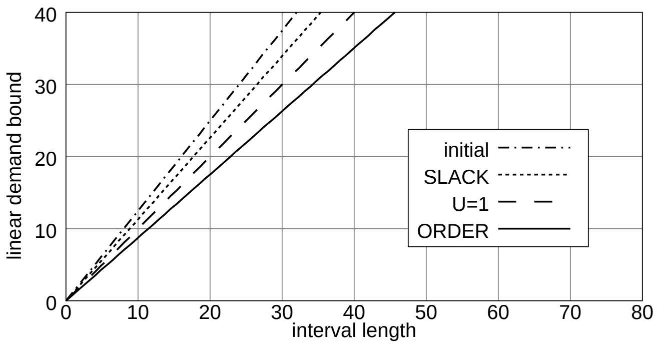

Figure 1: Bandwidth of pipelines.

Figure 2, instead, shows all the details of the `dbf`s. We highlight that the plotting is fully automatized by the script files.

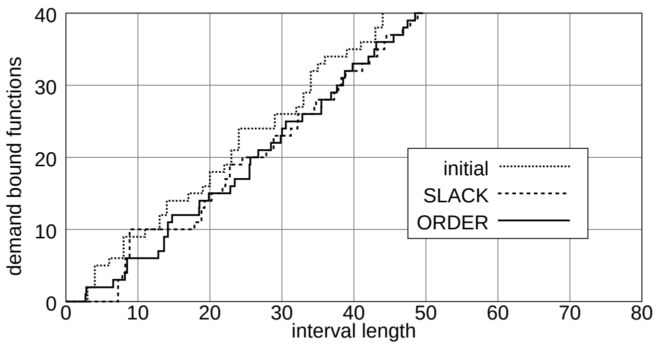

Figure 2: Bandwidth of pipelines.

REFERENCES

[1] K. Richter, R. Racu, and R. Ernst, "Scheduling analysis integration for heterogeneous multiprocessor SoC," in *Proceedings of the* 25th *Real-Time Systems Symposium*, Cancun, Mexico, Dec. 2003, pp. 236–245.

[2] W. Zheng, Q. Zhu, M. Di Natale, and A. Sangiovanni-Vincentelli, "Definition of task allocation and priority assignment in hard real-time distributed systems," in *Proceedings of the* 28th *IEEE Real-Time Systems Symposium*, Tucson, AZ, Dec. 2007, pp. 161–170.

[3] K. W. Tindell, A. Burns, and A. Wellings, "An extendible approach for analysing fixed priority hard real-time tasks," *Journal of Real Time Systems*, vol. 6, no. 2, pp. 133–152, Mar. 1994.

[4] N. Serreli, G. Lipari, and E. Bini, "Deadline assignment for component-based analysis of real-time transactions," in *2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Washington, DC, U.S.A., Dec. 2009.

[5] ——, "The demand bound function interface of distributed sporadic pipelines of tasks scheduled by edf," in *Proceedings of the 22nd ECRTS*, July 2010, pp. 187–196.

[6] S. K. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[7] R. Pellizzoni and G. Lipari, "Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling," *Journal of Computer and System Sciences*, vol. 73, no. 2, pp. 186–206, Mar. 2007.

[8] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings of the* 15th *IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, Dec. 1994, pp. 215–227.