Université libre de Bruxelles

Project 1
Global vs. Partitioned DM

Aldar Saranov, Przemyslaw Gasinski

Aldar.Saranov@ulb.ac.be
Przemyslaw.Gasinski@ulb.ac.be

INFO-F-404 Real-Time Operating Systems II (M-INFOS/F277)

Joël Goossens, Nikita Veshchikov

November 2016

Software description

Generator

Input/output

Command in form

```
./taskGenerator -u 70.0 -n 8 -o tasks.txt (-a 100.0)
```

-u – Total utilization in percent
-n – Number of tasks
-o – Output filepath/filename
-a – Approximate average WCET (optional argument, the higher it is, the more precisely the total utilization will be achieved. 100.0 by default if not specified)

Output is passed to the specified file. Every line is corresponds to a task. Order: Offset, Period, Deadline, WCET.

```
56 181 134 61
10 181 150 142
38 181 152 132
```

Generation algorithm

Generator uses following algorithm:

1) Calculate common period by formula $Period = 100 \times AverageWcet \times TaskNumber / TotalUtilisation$
2) For each task generate offset according to uniform distribution from some (hardcoded) $minOffset$ and $maxOffset$.
3) Distribute the ordered $Utilisation$ over each of $TaskNumber$ tasks.
   a. Set Min$Utilisation = \frac{UtilisationLeft \times (1 - Deviation)}{TaskLeftNumber}$.
   b. Set $MaxUtilisation = \frac{UtilisationLeft \times (1 + Deviation)}{TaskLeftNumber}$.
   c. If $MinUtilisation < 0$ $then$ $MinUtilisation = 0$
   d. If $MinUtilisation > 1$ $then$ $MinUtilisation = 1$
   e. Ensure that the rest tasks will get by 100% utilization each in worst case (extra constraint for $MinUtilisation$)
   f. Same (c) and (d) for $MaxUtilisation$.
   g. Generate utilization for the next task according uniform distribution from $MinUtilisation$ to $MaxUtilisation$.
   h. Calculate $UtilisationLeft$ and decrement $TaskLeftNumber$.
4) For each task calculate $Wcet_i = \frac{Period \times Utilisation_i}{100.0}$.
5) For each task generate $Deadline_i$ according to uniform distribution from $Wcet_i$ and $Period_i$.

Simulator

Input/output

```
./simDM [-g | -p] <tasksFile> <processorsNbr> (-s)
```

[-g|-p] – g corresponds to global strategy, p to partitioned.

<taskFile> - input tasks filename/filepath.
<processorNbr> - number of processors available during the simulation.
-s – optional argument, determines if the application will output the simulation steps one after another.

Outputs to console and to result.txt file following:

1. Minimum number of processors required for this input data (besides of the specified number of processors).
2. Flag if the scheduling is feasible for specified number of processors.
3. Simulation interval for every processor (they are equal in global case).
4. Total number of preemptions and per every processor.
5. Total number of idle time units and per every processor.
6. Total utilization and per every processor.

Before that it outputs to console the simulation steps one after another if "-s" was specified.

Simulation algorithm

We have two possible simulation algorithms which can process several events [1].

1. Future event list/set (similar to GPSS modeling language future event chain algorithm).
2. Delta T.

Future event chain has following algorithm (Figure 1). It is based on determining of the next event by comparing their planned time moments and executing the nearest event action with consequent transition of current time to that moment. It is effective at large time scale (which is our case) but slightly hard to implement due to obligation to control the future event list manually (add/remove/execute events).

Delta T has following algorithm (Figure 2). It is based on regular adding of a constant delta T timespan to the current simulation moment and analyzing which event actions have to be executed. It is more simple to implement and effective in cases when we have high possibility of obtaining events happening at the same moment. However this algorithm is not effective on large time scale due to regular status analyzing at every moment especially in cases if delta T is much less than average time intervals between expected events.

Eventually future event chain has been implemented due to its effectiveness. Events setup in future event list is illustrated in Table 1. Nonetheless we have modified end of simulation condition. Simulation is over when for every processor simulation time is over (first N events). The task events (start, finish, deadline) are grouped in three-event tuples.

Both in global and partitioned we have agreed on following event types:

0. Processor simulation over.
1. Job spawn.
2. Job finish.
3. Job deadline.

For 0-event type we have to:

1) Unassign processor from its task (if assigned).
2) Erase itself (set this event to unexpected).

For 1-event type we have to:

1) Check if previous deadline of this job was satisfied (otherwise fire "non-feasible").
2) Assign to the job time-left parameter to WCET of its task.
3) Set the next job spawn time to current time + task period.
4) Set the closest deadline to current time + task deadline.

For 2-event type we have to:

1) Set finish event to unexpected.
2) Set deadline event to unexpected.
3) Unassign task from its processor.

In addition after 0^{th}, 1^{st}, 2^{nd} type events we must execute reassignTask()

For 3^{rd} event type (if program got here) then it means that deadline was trespassed and the system is not feasible.

reassignTask() algorithm:

1) Assign tasks to processors and vice versa according to global or partitioned strategy.
2) Track preempted tasks and accumulate statistics.
3) Check that task does not migrate if it is not preempted in global case.
4) Check that task does not migrate at all in partitioned case.

Best fit

Was successfully implemented. Takes tasks and number of processors as arguments. Also returns unfeasible-flag in case if it not possible to distribute them.

Study

Study application does series of experiment tests. They are described in the test and analysis chapters.

Input/output

```
./study <input.txt> <output.txt>
```

<input.txt> - input filename/filepath
<output.txt> - output filename/filepath

Input file is shown in following way:

```
5

3
1
2
3

5
10
50
90
100
150
```

First number is the number of processors for each simulation. After that comes N (the number of task numbers for each experiment) with consequent N task numbers. After that comes M (the number of utilizations for each experiments) with consequent utilizations.

Output is written to the specified output file. Output is:

1. Utilizations read from the input.
2. Global load table.
3. Partitioned load table.
4. Global minimal processors table.
5. Partitioned minimal processors table.
6. Schedulable flag table.
7. Partitioned flag table.

Difficulties

Generation of fraction wcets

Initially it was proposed to fix the period (for each task) as a constant during task generation. The problem of such approach may be encountered if user sets too large value of $TaskNumber$ or too small $TotalUtilisation$. In such case wcets generated at the next step often were fractions between 0 and 1 and therefore were being truncated due to integer type. The actual utilization deviated from the ordered too much.

Overload of generated tasks

This is the reason why use step [3e] (in the algorithm description above). Suppose we have 3 tasks to which we have to assign 270% of utilization. If we neglect [3e] step then for some configuration we can obtain generation range [50%;100%]. In such case if we may generate the first task with 60% utilization then we will leave 210% for the rest 2 tasks which cannot distribute these utilization. We must have at least [70%;100%] generation range. $MinUtilisation$ should be not less than $UtilisationLeft - 100 \times TaskLeftNumber$ (if this expression is positive).

Too large hyper period

For random periods at large scale hyper period (which is LCM of periods) becomes even larger what leads to value overflow for big number of tasks. To resolve this problem the period has been set same (but different for different generations) for all the tasks. The hyper period becomes equal to this value without any harm caused to the simulation model.

Too large simulation timeline

Overflow bugs resolved by migrating from [unsigned int] to [long long].

Calculating the minimal number of processors required

Is calculated (Figure 3) using the number using a loop by regular incrementing the number of allocated processors with consequent simulation. If the number of processors exceeds the number of tasks then it returns non-feasibility value (since it will never be feasible in such case).

Test planning

The experiment space has 3 dimensions (1-type of strategy, 2-different utilizations, 3-different number of tasks). Number of processors is provided by user as fixed (this parameter does not affect minimal processors tables). Following test plan was performed:

1. Strategies: both.
2. Utilisations: 10%, 50%, 100%, 150%, 200%, 250%, 300%, 350%, 400%, 450%.
3. Numbers of tasks: 1, 2, 3, 5, 10.
4. Number of processors = 2.

Simulation analysis

Obtained results

Testing was conducting according to the high mentioned test plan. Results are shown in tables 2-7. In the tables system load was calculated even in cases of non-schedulability for the sake of

interest. “0” in partitioned load table means that non-schedulability was detected at the stage of task distribution over the processors and no simulation was carried out. “-1” in minimal processors tables means that such configuration in non-schedulable.

Based on this data some logical facts and correlations can be figured out:

1.

Conclusion

Appendix

Figures

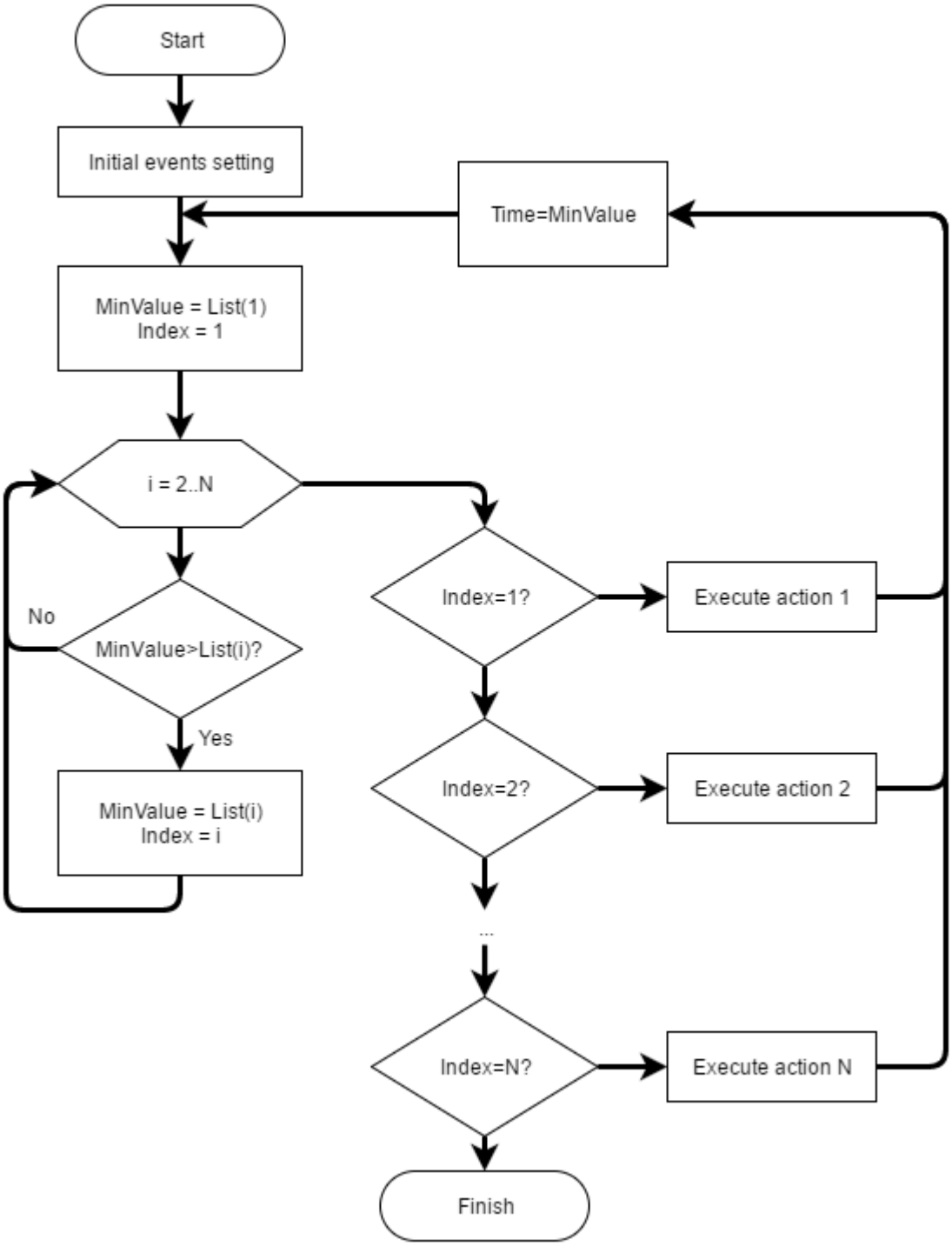

Figure 1. Future event list algorithm.

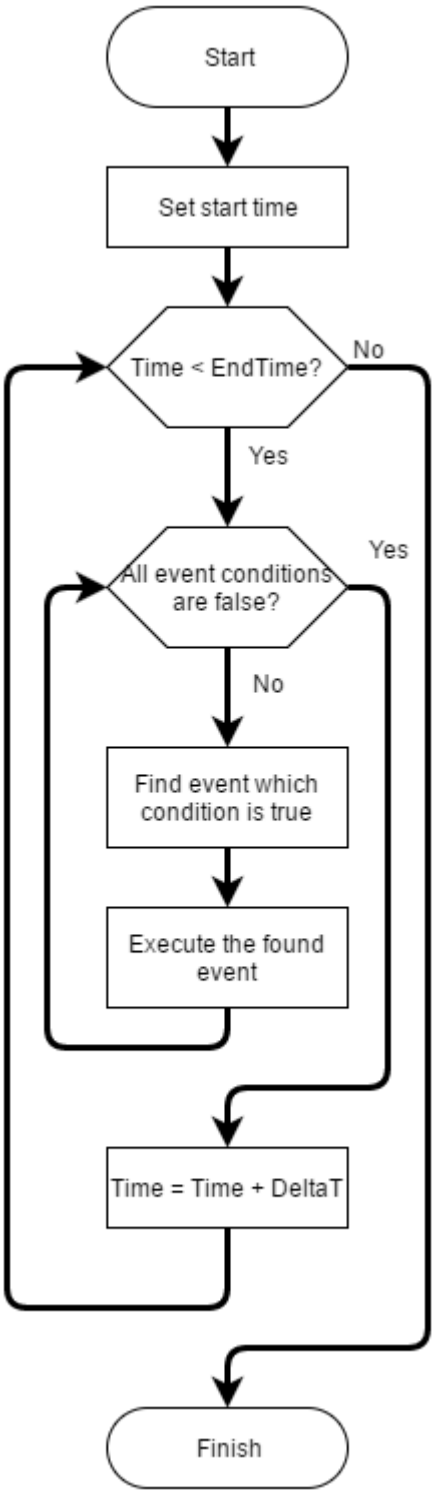

Figure 2. Delta T algorithm.

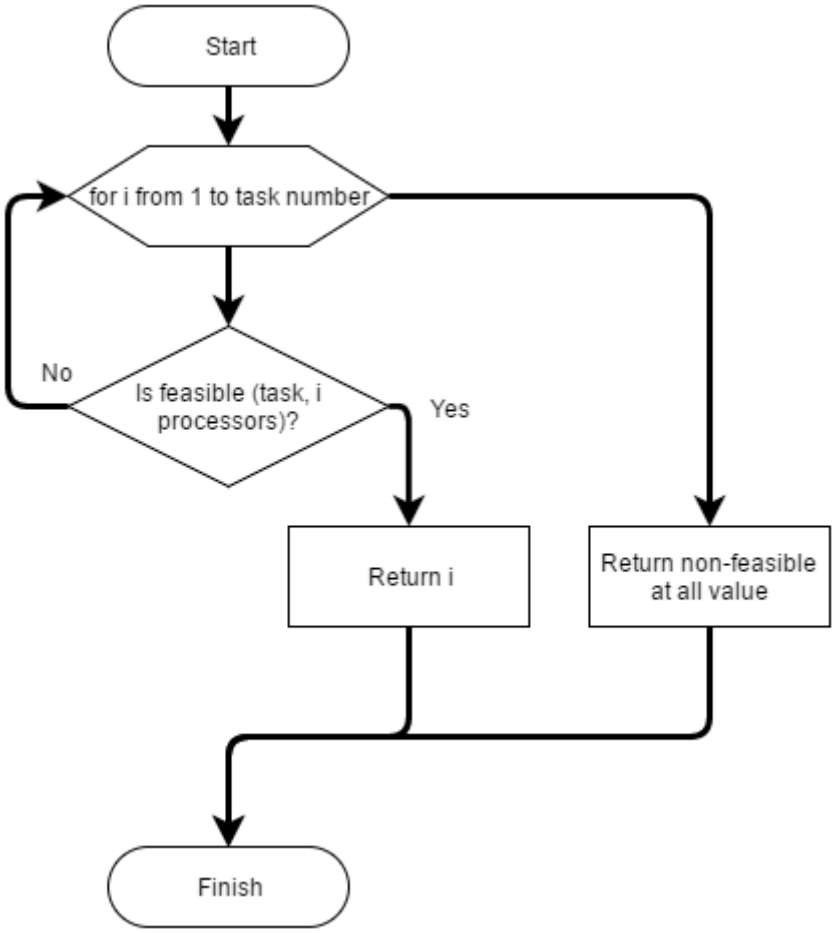

Figure 3. Minimum processor determiner algorithm.

Tables

Event index	Event meaning
0	Processor #0 simulation end
1	Processor #1 simulation end
2	Processor #2 simulation end
...	...
N-1	Processor #N-1 simulation end
N	Task #0 next job spawn
N+1	Task #0 expected finish
N+2	Task #0 next deadline
N+3	Task #1 next job spawn
N+4	Task #1 expected finish
N+5	Task #1 next deadline
...	...
N + M x 3 - 3	Task #M-1 next job spawn
N + M x 3 - 2	Task #M-1 expected finish
N + M x 3 -1	Task #M-1 next deadline

Table 1. Event setup illustration of future event list algorithm (N processors, M tasks).

Task num.\ Utilisation	10%	50%	100%	150%	200%	250%	300%	350%	400%	450%
1	0.099	0.488	0.956	0.88	0.847	0.816	0.785	0.756	0.735	0.709
2	0.107	0.513	0.975	1.395	1.779	1.393	1.28	1.181	1.084	0.981
3	0.109	0.530	1.024	1.485	1.915	2.319	2.706	2.234	2.142	2.04
5	0.107	0.527	1.028	1.504	1.961	2.396	2.817	3.214	3.604	3.762
10	0.117	0.576	1.128	1.659	2.168	2.658	3.128	3.577	4.018	4.439

Table 2. Test result. Load. Global.

Task num.\ Utilisation	10%	50%	100%	150%	200%	250%	300%	350%	400%	450%
1	0.097	0.445	0.803	0	0	0	0	0	0	0
2	0.101	0.478	0.775	1.346	1.773	0	0	0	0	0
3	0.104	0.492	0.966	1.502	1.852	2.267	2.690	0	0	0
5	0.103	0.505	0.958	1.725	2.292	2.447	2.680	2.975	3.548	3.905
10	0.103	0.506	0.986	1.621	2.157	2.548	3.141	3.5	3.956	4.429

Table 3. Test result. Load. Partitioned.

Task num.\ Utilisation	10%	50%	100%	150%	200%	250%	300%	350%	400%	450%
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	2	2	2	-1	-1	-1	-1	-1
3	1	1	2	2	3	3	3	-1	-1	-1
5	1	2	3	3	4	4	5	5	5	5
10	1	1	2	3	3	4	4	5	6	6

Table 4. Test result. Minimal processors. Global.

Task num.\ Utilisation	10%	50%	100%	150%	200%	250%	300%	350%	400%	450%
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	-1	2	2	-1	-1	-1	-1	-1
3	1	1	-1	-1	3	3	3	-1	-1	-1
5	1	1	-1	-1	-1	-1	-1	-1	5	5
10	1	1	-1	-1	-1	-1	-1	-1	-1	-1

Table 5. Test result. Minimal processors. Partitioned.

Task num.\ Utilisation	10%	50%	100%	150%	200%	250%	300%	350%	400%	450%
1	Yes	Yes	Yes	No	No	No	No	No	No	No
2	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
3	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
5	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
10	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 6. Test result. Minimal processors. Schedulable.

Task num.\ Utilisation	10%	50%	100%	150%	200%	250%	300%	350%	400%	450%
1	Yes	Yes	Yes	No	No	No	No	No	No	No
2	Yes	Yes	No	Yes	Yes	No	No	No	No	No
3	Yes	Yes	No	No	Yes	Yes	Yes	No	No	No
5	Yes	Yes	No	No	No	No	No	No	Yes	Yes
10	Yes	Yes	No	No	No	No	No	No	No	No

Table 7. Test result. Minimal processors. Partitioned.

References

1. https://books.google.ru/books?id=XUnPBAAAQBAJ&pg=PA29&lpg=PA29&dq=future+event+list+and+delta+t&source=bl&ots=Ws8uv5r1PN&sig=fpWhjoUvqLrKgaJYm2namMtCUZo&hl=ru&sa=X&ved=0ahUKEwimqcDUycLQAhUCqxoKHZeSDAcQ6AEIJTAB#v=onepage&q=future%20event%20list%20and%20delta%20t&f=false