OPT-*i*
An International Conference on
Engineering and Applied Sciences Optimization
M. Papadrakakis, M.G. Karlaftis, N.D. Lagaros (eds.)
Kos Island, Greece, 4-6 June 2014

SCHEDULING OF COMPUTATIONAL TASKS IN SWITCHED NETWORK-BASED IMA SYSTEMS

Vasily V. Balashov[1], **Vadim A. Balakhanov**[1], **and Valery A. Kostenko**[1]

[1] Department of Computational Mathematics and Cybernetics,
Lomonosov Moscow State University
Leninskie Gory, MSU, 1, Bldg. 52, Room 764, Moscow, Russia
e-mail: {hbd, baldis, kost}@cs.msu.su

Keywords: Real-Time Systems, Computations Scheduling, Network Load Optimization.

Abstract. *In this paper, the problem of automatic scheduling of computational tasks in integrated modular avionics (IMA) systems is addressed. This problem breaks down into: (1) assigning the partitions (groups of related tasks) to processor cores of different IMA modules, and (2) constructing sets of task execution windows for all cores, along with defining the tasks' priorities. For the first subproblem, a greedy algorithm which minimizes the inter-module data exchange is presented. For the second subproblem, an algorithm is presented which constructs a schedule conforming to a set of constraints imposed by the IMA architecture and specifics of the target system. Results of experimental evaluation of these algorithms are presented. The algorithms are implemented in a tool system for scheduling of computations in IMA systems.*

1 INTRODUCTION

Integrated Modular Avionics (IMA) systems are distributed real-time systems consisting of computing modules connected by a communications network. A typical IMA system utilizes several types of unified modules and a switched network (fabric) supporting virtual data transfer channels. Each module contains one or more multicore processors. The network is based on either AFDX or Fibre Channel technology.

Computational tasks for IMA systems are grouped into *partitions* according to their criticality and data coupling. All tasks from a partition are executed on the same processor core. Processor time is allocated to each partition in the form of *execution windows*. The notions of partition and execution window used in this paper are described in detail by the ARINC 653 standard [1].

A task set for a modern IMA system includes several hundreds of periodic tasks. Considering different periods of tasks, the number of jobs in the scheduling interval (also known as the major frame of the schedule) reaches several thousand. Tasks running on different modules generate traffic in the inter-module network. To ensure scalability of the system, this traffic should be minimized. Meanwhile the utilization of processor cores in the modules should be balanced, and task execution schedules must conform to the constraints imposed by IMA architecture and the specifics of the target system. This makes the scheduling problem for IMA systems sufficiently complex and calls for development of scheduling algorithms ant tools for this problem

Existing approaches (e.g. [2, 3, 4, 5]) for scheduling in time-partitioned systems deal with either too abstract or too constrained description of the target system, and can hardly be improved to solve the real-life scheduling problem described below.

In this paper we present algorithms for scheduling the computation tasks in RMA systems, including partitions' assignment to processor cores and construction of tasks execution window for the cores. The rest of the paper is structured as follows. Section 2 describes the organization of computations in an IMA system from the scheduling point of view. Section 3 presents the scheduling problem statement, structure of the solution and constraints on the solution. Section 4 contains an overview of several existing approaches to scheduling in time-partitioned systems, as well as analysis of these approaches' applicability to the problem under consideration. In the Section 5 the proposed scheduling algorithms are described, and results of their experimental evaluation are presented. Section 6 briefly presents a tool system in which the algorithms are implemented. In the conclusion, directions for future work are outlined.

2 ORGANIZATION OF COMPUTATIONS IN IMA SYSTEMS

2.1 IMA system structure

In this paper we consider IMA systems consisting of multiple computing modules, each of which contains one or more multicore processors. Different types of processors can be used in the IMA system, so execution time for the same task may vary depending on which processor's core the task is running on.

The modules are connected by a switched network which implements virtual channels. The traffic in one virtual channel is considered isolated from the traffic in any other virtual channel. This is the most important feature of such a switched network from the scheduling point of view, as it enables independent accounting for transfer delays of different messages.

2.2 Structure of the task set

The task set for the IMA system consists of periodic tasks grouped into partitions according to logical relationship and data coupling. The tasks take input data and generate output data in the form of messages, so the set of periodic tasks is complemented by a set of messages.

A message between two tasks constitutes a data dependency. If both tasks have same period, the dependency is considered synchronous, so a job ("instance") of the receiver task must wait for arrival of the message from the corresponding job of the sender task in order to start. If two tasks have different periods, there is no requirement that the receiver job waits for the message from the sender job, so the dependency is asynchronous. Further in this paper synchronous dependencies between tasks are mainly considered.

Each task within a partition has a priority which is unique within the partition. Priorities of tasks belonging to different partitions are incomparable.

2.3 Timings of message transfer and task execution

In this paper we assume that for each message between tasks running on different modules there is a dedicated virtual channel in the switched network. In this case, transfer time of the message is not affected by the network load, and can be considered constant. In fact, in AFDX and Fibre Channel networks with virtual channels there is certain influence of traffic in some virtual channels on transfer timings for messages in other virtual channels, but this influence is insignificant in comparison to, for instance, common buses like CAN [6]. There are several techniques to provide safe estimation of data transfer timings for a switched network with virtual channels, for instance [7, 8]. We assume that one of these techniques is used to calculate the maximum transfer time for each message. These transfer times are considered input data for the scheduling problem.

If sender and receiver tasks belong to different partitions running on the same module, the messages between these tasks are transferred through the module's memory. So for every message between tasks from different partitions, two timings are given: one for transfer through the network, and the other (usually shorter) for transfer through the memory.

Message transfers between tasks from the same partition are considered instant, because tasks of a single partition are bound to the same processor core and exchange messages through shared memory.

For the tasks from the task set, worst case (maximum) execution times are given for each processor type in the IMA system.

2.4 Scheduling scheme

Processor time is allocated to partitions by means of execution windows. An execution window is a time interval within which the tasks of a specific partition can be executed on the processor core to which the partition is assigned. Execution windows on the same processor core cannot overlap. The grid of execution windows is the same for all cores of the same module.

The schedule for the IMA system defines the set of execution windows (for each module) and assignment of partitions to the execution windows (for each core). The schedule is constructed for the duration l_{int} of the scheduling interval, which equals the least common multiple of the tasks' periods. This schedule is constructed statically and loaded into the IMA system as a part of its configuration.

The sequence (or schedule) of tasks execution is determined by the scheduler of the IMA operating system during the IMA system runtime, according to the schedule of execution windows, partitions binding to the windows, and task priorities.

In the IMA system runtime, after the end of the scheduling interval, the schedule of execution windows is executed again.

Within a window, tasks of the partition which are ready for execution start according to their priorities. The task is "not ready" if, for instance, some of its synchronous input messages have not yet arrived, or the successive iteration of the task period has not yet begun. After the task execution is complete, transfer of all its output messages begins. If a task starts within some window, but is not finished before the end of the window, its execution resumes in the next window of the same partition, again according to the task priority.

The execution of the tasks is preemptive.

As the tasks in the task set generally have different periods, the static scheduling algorithms operate with jobs (instances of the tasks). For each periodic task there is a set of jobs. Deadline interval for a job is defined by the task period and number of task iteration which corresponds to the job. For instance, if T is the task period, i is the number of task iteration (numbering starts from 1), then the deadline interval for the job is $[(i-1)*T; i*T]$. The number of jobs for a task during one scheduling interval equals $\lceil l_{\text{int}} / T \rceil$.

For chains of jobs with synchronous data dependencies, finish deadlines of jobs can be refined: a job must finish (and produce its output messages) early enough to allow the dependent jobs to finish within their deadlines at least in the best case, when these jobs are started as soon as possible.

3 THE PROBLEM OF COMPUTATIONS SCHEDULING IN IMA SYSTEMS

3.1 Problem statement

The computations scheduling problem for IMA systems is stated as follows.

Input data:

Description of the IMA system:

- set of computational modules in the IMA system;
- for each module: number and types of processors;
- set of processor types;
- for each processor type: number of cores (all cores of the same processor are uniform), window initialization duration, context switch duration;
- for each processor core: upper limit on utilization.

Description of the workload (tasks, messages):

- set of tasks;
- for each task: period, worst case (maximum) execution time for every processor type, initial priority;
- set of partitions;
- for each partition: set of tasks belonging to the partition (these sets do not intersect), set of processor cores to which the partition can be assigned;
- optional, for some partitions: fixed assignment to processor cores (essential for incremental development of an IMA system);

- set of messages;
- for each message: sender task, receiver task, size (bytes);
- for each message between tasks from different partitions: maximum duration of transfer through the network, maximum duration of transfer through memory.

Note: initial priorities for the tasks of a partition may be defined only for some of the tasks and are not necessarily unique within the partition. These priorities define only a partial ordering of the tasks. During schedule construction, final unique priorities are calculated for all the tasks of the partition. Final priorities define the complete ordering of tasks within the partition.

Output data (solution):

- assignment of partitions to processor cores;
- for each module: set of execution windows;
- for each window: start time; finish time; partition, the tasks of which are executed in the window (there can be windows without partitions, e.g. on cores to which no partitions are assigned);
- for each task: its final priority.

Criterion to be minimized: total size of messages transferred through the network during the scheduling interval.

Note: the reason for choosing this criterion is that the size of messages is an additive characteristic of network load. To the opposite, the message transfer durations are not additive as the virtual channels in the network generally have different throughput limits, so a message of the same size has long transfer duration for a "slow" virtual channel and short transfer duration for a "fast" virtual channel.

The chosen criterion can be minimized by assigning intensely interacting (via message transfer) partitions to the cores of same modules, while meeting the limits on core utilization.

Balancing the utilization between processor cores can be achieved by setting proper limits on utilization for the processor cores. Introducing "balanced utilization" as the second criterion, analyzing and solving the resulting bi-criteria problem can be considered future work.

We do not present the strictly formal statement of the scheduling problem here, as it requires an extensive set of notations to represent all the involved objects and constraints on these objects.

3.2 Constraints on the solution

A correct solution for the scheduling problem must meet a number of constraints, including:

- constraints on execution windows:
  - the set of execution windows must be the same for all cores of the same module;
  - execution windows for a specific processor core must not overlap;
  - no more than one partition can be bound to any window;
- constraints on partitions assignment to processor cores:
  - each partition must be assigned to no more than one processor core;
  - each partition can be assigned only to processor cores which are permitted for this partition in the input data set;
  - for each core, the specific upper limit on utilization must not be exceeded;
- constraints on final priorities:
  - for each partition, final priorities of its tasks must be unique;

- constraints on the sequence of jobs execution:
  - each job must start and finish within its deadline interval;
  - a job within a window can start (or resume) its execution no earlier than <*beginning of the window*> + <*window initialization duration*>;
  - if the partition bound to the current window is different than the partition bound to the previous window on the same core, than a job within the current window can start (or resume) its execution no earlier than <*beginning of the window*> + <*window initialization duration*> + + <*context switch duration*>;
  - a job cannot start if any of its synchronous input messages have not yet arrived;
  - jobs which are ready to start execute according to their priorities, with preemption enabled.

The latter group of constraints applies to the schedule of jobs execution. This schedule is not a part of the solution, as the IMA system constructs it in runtime and does not take it as input data. So, to check the correctness of the solution, one has to simulate the IMA system run (from the dynamic scheduler's point of view) and construct the schedule of jobs execution for the duration of a single scheduling interval. To check the pessimistic case, all job execution times are considered worst-case during the simulation.

4 RELATED WORK

4.1 Scheduling tools and algorithms for systems with time-partitioned scheduling

There are a number of papers describing scheduling tools and/or algorithms targeted at systems with time-partitioned scheduling, which operate more or less close to the ARINC 653 standard. However the scheduling problems formulated in these papers substantially differ from the one described above.

In the paper [2] tasks are not considered as separate entities. The workload is defined as a set of partitions. Each partition has a period, and execution windows for the partition occur in the schedule with this period. No data dependencies between partitions are taken in account. The scheduling problem is formulated as a constraint satisfaction problem and a constraint logic programming approach is used to solve it.

The level of detail for the target system description in [2] is not suitable for the scheduling problem stated in Section 3.1. Lack of support for data dependencies makes the scheduling approach proposed in [2] even less usable for real IMA systems.

In [3] the target system is represented in more detail. A set of tasks for the system is defined, with priority, period, execution time, deadline interval and release jitter for each task. Data dependencies between tasks are not explicitly specified; the authors suggest that deadline intervals should be constructed to reflect the dependencies. For each partition, its share in CPU utilization is specified, as well as its period. The periods are aliquant, i.e. for each pair of periods one of them is a multiple of the other. The set of execution windows is constructed according to Earliest Deadline First discipline (the algorithm is not explicitly described).

The scheduling problem stated in [3] assumes more regular operation of the target system than the problem considered in our paper (e.g. aliquant periods). Lack of explicit consideration of inter-task dependencies makes the scheduling approach proposed in [3] hard to apply to the problem stated in Section 3.1; in fact, to determine the deadline intervals which take data dependencies in account, a significant sub-problem of the scheduling problem needs to be solved. In our paper we propose an algorithm which concurrently constructs a schedule of

tasks execution, refines the deadlines of remaining tasks according to data dependencies, and builds the set of execution windows.

It should also be noted that in [3] the priorities of the tasks are given as input data, and no approach to priority assignment is proposed.

In [4] an original scheduling approach is presented, in which the tasks are treated as parts of “end-to-end flows” (ETEF) which are sequences of tasks. Such flow is the main subject for scheduling; deadlines are defined for ETEFs instead of tasks. The Xoncrete tool described in [4] calculates “best” periods for ETEFs in order to minimize their least common multiple (i.e. the duration of the scheduling interval). The periods of tasks are derived from the calculated periods of ETEFs.

The scheduling approach proposed in [4] is not suitable for solving the problem stated in Section 3.1, as the tasks periods are fixed in the input data for this problem. No details on the algorithm for construction of execution windows are provided in [4].

The scheduling algorithm proposed in [5] is based on ant colony approach. This algorithm assumes a fixed binding of tasks to execution windows. Such assumption is not valid for the IMA system scheduling logic described in Section 2.4: in our case, the actual subset of partition’s tasks to be executed in a given window depends on execution timings of previous tasks during a specific run of the IMA system. The assumption mentioned above is crucial for the logic of algorithm proposed in [5] and there is no easy way to evolve this algorithm to support IMA systems for which this assumption is invalid.

4.2 Schedulability analysis tools

The tools RAPID RMA [9], Cheddar [10], MAST [11] provide support for real-time scheduling analysis. These tools support complex dependencies between tasks, as well as several scheduling disciplines. Support for time-partitioned scheduling, including ARINC 653 compliant version, is either available in current version of the tools, or can be added via plugins.

The tools mentioned above can serve as independent verification tools for the schedules of task execution windows constructed by the algorithms and a tool system described in this paper. Integration with one of these tools can be considered future work.

5 PROPOSED SCHEDULING ALGORIGHMS

5.1 Problem decomposition

The problem stated in Section 3 can be solved in two steps:

1. Assigning the partitions to processor cores of the modules. On this step, the traffic on the network is to be minimized (it should be noted that message transfer between partitions on the same module does not utilize the network). For each processor core, a specific upper limit on utilization must not be violated.

2. Constructing the sets of execution windows for all processor cores of the IMA system. On this step, final priorities of the tasks must also be determined.

The assignment of partitions to processor cores is a part of data necessary for scheduling of execution windows. Therefore we propose to perform the steps 1 and 2 sequentially. More sophisticated approaches exist, e.g. an iterative one where unsuccessful execution of step 2 provides some feedback to the next iteration of step 1. Exploration of such approaches is considered future work.

5.2 Algorithm for assigning the partitions to processor cores

This algorithm assigns partitions to processor cores of the computational modules of the IMA system.

The algorithm minimizes the criterion introduced in Section 3.1, i.e. the total size of messages transferred through the network through the duration of the scheduling interval. Minimization of the criterion is achieved through assignment of intensely interacting (via message transfer) partitions to the cores of same modules, while meeting the limits on core utilization. The algorithm implements greedy strategy.

The following notations are used to describe the algorithm.

- $v^C(p,q)$ – total size of messages transferred between the partitions p and q through the duration of scheduling interval;
- $v^C(p)$ – total size of messages sent and received by the partition p through the duration of the scheduling interval;
- $v^C(p,Q)$ – total size of messages transferred between the partition p and partitions from the set Q through the duration of the scheduling interval;
- $\hat{v}^C(p,m)$ – total size of messages transferred between the partition p and partitions *not* assigned to the module m through the duration of the scheduling interval;
- P – set of all partitions;
- P' – set of partitions already assigned to processor cores. At the start of the algorithm, this set is non-empty if for some partitions the assignment to processor cores is fixed in the input data.

The scheme of the algorithm is as follows.

1) choose a partition for assignment from the set $P \setminus P'$:
   if P' is empty, or $v^C(p,P') = 0$ for all partitions $q \in P \setminus P'$
   then choose the partition p with maximum value of $v^C(p)$;
   else choose the partition p with maximum value of $v^C(p,P')$;
2) sort the modules by increase of $\hat{v}^C(p,m)$;
3) in a loop through the modules:
   a) if the current module m includes a core to which the partition p can be assigned without exceeding the utilization limit, then assign p to this core (if there are several such cores on the module, choose a core with maximum remaining margin to the utilization limit); go to step 4;
   // *in case the is no such core in the current module*
   b) if the partitions on the current module m can be reassigned to cores of m so that one of the cores is sufficiently offloaded to assign p to it without exceeding the utilization limit,
   then perform the reassignment; assign p to that core; go to step 4;
   Note: partitions for which the assignment to the cores is fixed in the input data can not be reassigned.
   // *in case the partition cannot be assigned to a core of the current module*
   c) continue the loop;
4) if no core was chosen on step 3, then **stop** (unsuccessful completion);

5) if the set P' is not empty, go to step 1, else **stop**.

Choice of this greedy scheme is motivated by the following factors:

- low computational complexity even for large numbers of partitions and cores;
- good results on available data for IMA systems (see Section 5.4 for details);
- ability to modify a partially defined assignment of partitions on step 3, allowing to decrease the risk of construction of a suboptimal solution because of decisions made on the early steps of algorithm's execution (this risk is typical for a "pure" greedy scheme).

This algorithm is finite, because on every iteration of the loop it either assigns a partition to a core (the number of partitions is finite), or stops.

Computational complexity of this algorithm depends on the numbers of partitions (N_P), modules (N_M), cores (N_C), tasks (N_{task}) and messages (N_{msg}).

The order of the algorithm's complexity is: $N_P*(N_P*N_M+N_C)+N_{msg}+N_{task}$.

An alternative for the algorithm described above can be an algorithm based on the container scheme. The problem of assignment of partitions to processor cores can be considered as a problem of packing objects into containers. The containers correspond to cores; container's remaining capacity is the utilization margin of the core, i.e. the difference between the utilization limit and the current utilization. The objects correspond to partitions; object's volume is the partition's contribution to the core utilization; object's cost is the part of traffic which becomes "internal" for the module when the object is placed in the container (i.e. the partition is assigned to a core).

This statement of the packing problem substantially differs from the traditional statement (with fixed volumes and costs of objects):

- object's volume depends on the choice of container, as the task execution times differ among processor types;
- object's cost depends on which other objects are packed in the same container.

Therefore, the existing algorithms for solving the container packing problem require modifications in order to solve the described problem. Solving of the partitions assignment problem as a container packing problem can be considered future work.

5.3 Algorithm for construction of the set of execution windows

The input data set for this algorithm is the input data set for the scheduling problem (see Section 3.1) plus the assignment of partitions to processor cores.

The general idea of the algorithm is to construct a temporary multi-processor static schedule of jobs execution (*jobs schedule*) for the duration of the scheduling interval and, in parallel, determine the bounds of the execution windows and the final priorities of the tasks.

The jobs schedule is constructed concurrently for all processor cores of the IMA system. For each *i*-th core (in a continuous numbering of all processor cores), a "current time" counter t_i is maintained. Each t_i starts from *0* and can only grow through the algorithm's execution. Jobs for the *i*-th core can only be scheduled at t_i or later.

A job for *i*-th core (i.e. belonging to a partition assigned to this core) is considered *ready* if:

- t_i belongs to the job's deadline interval;
- all synchronous messages for this job have arrived.

On each iteration the algorithm processes the core with minimum t_i. For this core the algorithm chooses a job fitting the following conditions:

- the job is ready;
- if initial priority is defined for the job: there is no other ready job with higher initial priority from the same partition;
- if initial priority is not defined for the job: there is no other ready job without initial priority from the same partition, such that the first job of its task precedes the first job of the task to which the job under consideration belongs.

If there are several jobs fitting these conditions, the algorithm chooses one of them which has the minimum finish deadline.

If a job is chosen, the algorithm schedules it on the current core. If the job belongs to a different partition than the previous job scheduled for the same core, a new window is opened (the current window is closed), and the job starts in it after the duration of window initialization and context switch. On all other cores of the same module, new windows are also opened (and current ones are closed), but no context switch is performed. Finish times of jobs in the middle of which a window change occurs are increased by window initialization duration. It should be noted that window initialization without a context switch is a cheap operation in IMA systems.

After a job is scheduled, arrival times for all its output messages are calculated (start deadlines of the dependent jobs are corrected accordingly); t_i is shifted to the job's finish time, and the algorithm starts a new iteration.

If no job is chosen and scheduled on an iteration (this means that no jobs are ready for the *i*-th core), the algorithm shifts t_i to the minimum time at which a ready job will be available at any core, *plus* one minimum time increment if that job is from another core.

The jobs that failed to be scheduled within their deadline intervals are moved to the set of unscheduled jobs.

After the jobs schedule is constructed, final priorities for the jobs are calculated independently for all partitions. The scheme for final priorities calculation is as follows:

1) tasks without initial priorities are ordered based on the order of first occurrences of these tasks' jobs in the jobs schedule;
2) tasks with initial priorities are ordered based on the values of the initial priorities; if initial priorities of some tasks are equal, these tasks are ordered based on the order of first occurrences of these tasks' jobs in the jobs schedule;
3) orderings 1) and 2) are merged to produce a total ordering of tasks; final priorities are assigned according to this ordering.

This algorithm is finite, because on every iteration it increases the current time counter t_i for one core (the increment is no less than a certain minimum), and the maximum value for t_i is finite, namely l_{int}.

Computational complexity of this algorithm depends on the numbers of partitions (N_P), modules (N_M), cores (N_C), tasks (N_{task}), messages (N_{msg}) and jobs (N_{job}).

The order of the algorithm's complexity is: $N_{job} * (N_{task}^2 + N_C * \ln N_C + N_{task})$.

The algorithm described above does not take any measures against creating series of "too small" windows in case context switches on different cores of one module densely follow

each other in some time frame. To avoid creating series of small windows, as well as to make the set of windows more regular, an additional constraint can be introduced that window open and close times must be multiples of a given value.

We suppose that the IMA system scheduler which executes the task set according to the constructed set of windows and assignment of task priorities will operate so that the dynamically constructed jobs schedule will be close to the one constructed a priori by the algorithm described above. To check this, a verification procedure is always run after the scheduling algorithm. This procedure simulates the tasks execution and message transfer in the IMA system (only timings and scheduling logic are simulated), and checks that all tasks meet their deadlines. A similar approach for solution verification is used in [12].

5.4 Experimental evaluation of the scheduling algorithms

The proposed scheduling algorithms were evaluated from the following points of view:

- practical applicability to a real IMA system;
- quality of the partition assignment algorithm results.

Note: of the two proposed algorithms, only the partition assignment algorithm has an explicitly defined optimized criterion and thus the quality of its results can be evaluated. By quality of a solution we mean the deviation of the solution from the optimum one by value of the optimized criterion. The second algorithm's aim is to produce a solution that meets all the imposed constraints.

To check practical applicability, the algorithms were executed on a full task set for a real IMA system. This data set includes 9 partitions with a total of 164 periodic tasks. The tasks produce 163 messages. The frequencies (and thus periods) of the tasks are generally not aliquant, e.g. there are tasks with frequencies of 12.5 Hz and 10 Hz. Highest frequency of a task is 100 Hz, and the lowest frequency is 1 Hz.

The IMA system contains 3 modules with several processor cores on each module. The pair of proposed algorithms successfully assigned the partitions to cores, produced the set of execution windows and set of tasks' priorities. The verification procedure confirmed that all the jobs meet their deadlines on a simulated IMA system.

Total execution time of two algorithms and the verification procedure was less than one minute on an AMD E-450 1.5 GHz processor.

To check the quality of the partition assignment algorithm results, the real task set was simplified to enable running an exact algorithm based on branch-and-bound scheme in reasonable time (approximately 10 minutes; the greedy algorithm finishes in less than 1 second). The number of partitions was reduced to 5, and the number of modules to 2. The task execution times were randomly modified (+/- 50% to the original time) in order to produce three test task sets. The processor types on module 1 and module 2 were made different, with different task execution times.

For each set of workload (partitions, tasks, messages), experiments were run with different limits on core utilization, e.g. for the first data set: 0.6/0.6 (for each core of module 1 and each core of module 2 respectively), 0.5/0.5, 0.6/0.4, 0.5/0.4.

In 10 of 12 experiments, the greedy algorithm produced the same solution as the exact one, i.e. the optimum solution.

In one experiment, the minimized criterion (see Section 3.1) on the solution from the greedy algorithm was 20% greater (worse) than on the optimum solution.

In another one experiment, the greedy algorithm failed to find a solution, meanwhile the exact algorithm found one. Detailed examination revealed that due to strict limit on core utilization in this experiment, the optimum solution was the only existing correct solution. The

solution partially constructed by the greedy algorithm completely matched the exact solution except for assignment of one partition to a "wrong" module. In practice, slight relaxation of core utilization limit (+2% on one core of that module) enabled the greedy algorithm to produce a solution which is close to the optimum one.

6 TOOL SYSTEM FOR COMPUTATIONS SCHEDULING IN IMA SYSTEMS

The algorithms proposed in this paper are implemented in a tool system for scheduling of computation in IMA systems. The tool system has a modular structure (see Figure 1) derived from the architecture of the system for scheduling of data exchange on channels with centralized control [13]. The system is accepted for experimental operation by one of the leading Russian aircraft design companies.

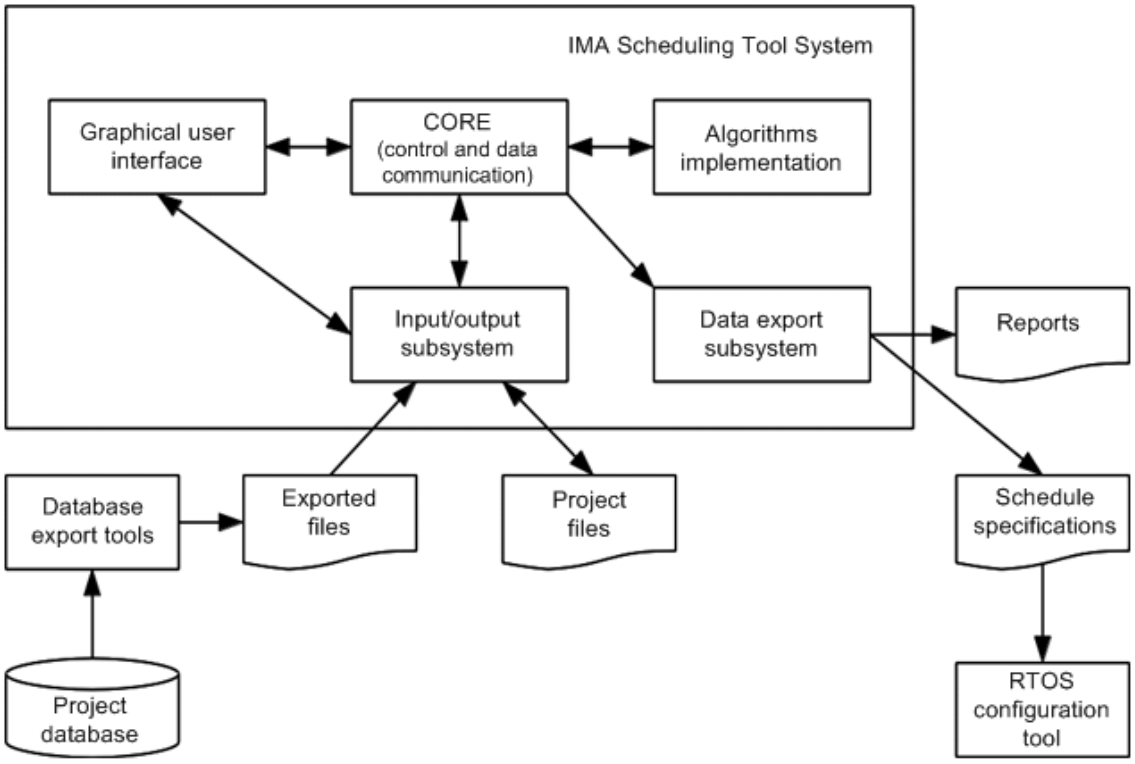

Figure 1: Structure of the tool system.

7 CONCLUSIONS AND FUTURE WORK

In this paper a problem statement and algorithms were presented for scheduling of computations in integrated modular avionics (IMA) systems based on switched network with support virtual channels. Besides construction of a set of task execution windows, the proposed algorithms perform assignment of partitions to processor cores (with network load minimization) and determine the priorities for the scheduled tasks. The algorithms are implemented in a tool system which is accepted for experimental operation.

The problem statement and the proposed algorithms deal with more detailed and realistic specification of an IMA system and its workload than the existing algorithms described in the overview of related work.

Future work on the computations scheduling algorithms and tools for IMA systems includes:

- exploration of the scheduling problem stated as a bi-criteria problem with "balanced core utilization" as the second optimized criterion;

- exploration of iterative approaches in which an unsuccessful schedule construction attempt provides feedback for correcting the assignment of partitions to processor cores;
- formulation of the partitions to cores assignment problem as a container packing problem and development of container packing algorithms to solve this problem;
- development of scheduling algorithms for IMA systems based on a shared common bus [14], in which case message transfers may cause conflicts on the bus;
- integration of the scheduling tool system with an existing IMA system simulation tool to provide independent verification of constructed schedules;
- improving the tool system and algorithms according to feedback from experimental operation.

REFERENCES

[1] "ARINC 653 – An Avionics Standard for Safe, Partitioned Systems". *Wind River Systems / IEEE Seminar*. August 2008. [http://www.computersociety.it/wp-content/uploads/2008/08/ieee-cc-arinc653_final.pdf]

[2] Goltz, H.-J., Pieth, N. A Tool for Generating Partition Schedules of Multiprocessor Systems. *Proc. 23rd Workshop on (Constraint) Logic Programming*, Potsdam, Germany, 2009.

[3] Easwaran A., Lee I., Sokolsky O., Vestal S. A Compositional Framework for Avionics (ARINC 653) Systems. *Proc. IEEE Real-Time Computing Systems and Applications 2009*, Beijing, Aug 24-26, 2009.

[4] Brocal V., Masmano M., Ripoll I. et al. Xoncrete: a Scheduling Tool for Partitioned Real-Time Systems. *Proc. ERTSS-2010*, Toulouse, France, 2010.

[5] Kostenko V. A., Plakunov A. V. An Algorithm for Constructing Single Machine Schedules Based on Ant Colony Approach. *Journal of Computer and Systems Sciences International*, **52** (6), 928–937, 2013.

[6] Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling. *ISO 11898-1:2003 Standard*, ISO, 2003.

[7] Bauer H., Scharbarg J. L., Fraboul C. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. *Proc. Emerging Technologies & Factory Automation,* 1-8, 2009.

[8] Gutiérrez J. J., Palencia J. C., Harbour M. G. Response time analysis in AFDX networks with sub-virtual links and prioritized switches. *Proc. JTR'12*, 2012.

[9] RAPID RMA: The Art of Modeling Real-Time Systems. *Obtained through Internet*: http://www.tripac.com/rapid-rma, [accessed 10.04.2014]

[10] The Cheddar project: a free real time scheduling analyzer. *Obtained through Internet*: http://beru.univ-brest.fr/~singhoff/cheddar, [accessed 10.04.2014]

[11] MAST: Modeling and Analysis Suite for Real-Time Applications. *Obtained through Internet*: http://mast.unican.es/, [accessed 10.04.2014]

[12] Kostenko V.A. Scheduling Algorithms for Real-Time Computing Systems Admitting Simulation Models. *Programming and Computer Software*, **39** (5), 255–267, 2013.

[13] Balashov V. V., Balakhanov V. A., Kostenko V. A., Smeliansky R. L., Kokarev V .A., Shestov P. E. A technology for scheduling of data exchange over bus with centralized control in onboard avionics systems. *Proc. Institute of Mechanical Engineering, Part G: Journal of Aerospace Engineering*, **224** (9), 993–1004, 2010.

[14] KSHES// Shestov P. E., Kostenko V. A., Balashov V. V. Scheduling Problems In Embedded Real-time Systems. *Proc. 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems (PDeS 2012)*, 302–306, 2012.