2014 43rd International Conference on Parallel Processing

Analysis and Design of Fault-Tolerant Scheduling for Real-Time Tasks on Earth-Observation Satellites

Xiaomin Zhu, Jianjiang Wang, Ji Wang
Science and Technology on Information
Systems Engineering Laboratory
National University of Defense Technology
Changsha, China 410073
Email: {xmzhu, jianjiangwang, wangji}@nudt.edu.cn

Xiao Qin
Department of Computer Science and
Software Engineering, 3101E Shelby Center
Auburn University
Auburn, AL 36849-5347, USA
Email: xqin@auburn.edu

***Abstract*—Fault-tolerant scheduling is an efficient approach to improving the reliability of multiple earth-observing satellites especially in some emergent scenarios such as obtaining photographs on battlefields or earthquake areas. Unfortunately, little work has been done to deal with the fault-tolerant scheduling on satellites. To address this issue, this paper presents a novel dynamic fault-tolerant scheduling model using primary-backup policy to tolerate one satellite's permanent failure at one time instant. On this basis, we propose a novel fault-tolerant satellite scheduling algorithm named FTSS, in which an overlapping technology is adopted to improve the resource utilization. Besides, the FTSS employs the task merging strategies to further enhance the schedulability. To demonstrate the superiority of our FTSS, we conduct extensive experiments by simulations using real-world satellite parameters from STK to compare FTSS with other baseline algorithms. The experimental results indicate that FTSS efficiently improves the scheduling quality of others and is suitable for fault-tolerant satellite scheduling.**

I. Introduction

Earth observation satellites (EOSs) are platforms equipped with optical instruments that orbit the Earth to take photographs of specific areas at the request of users [1]. As the number of satellites grows, multiple satellites are commonly designed to operate in a coordinated way to perform a mission, which significantly improves system performance, cost, and survivability compared with the traditional single-satellite deployments [2]. Noticeably, real-time applications have been developed and deployed on multiple EOSs, in which the correctness depends not only on the logical results (photographs imaged by EOSs), but also on the time instants at which these results are produced. For example, when an earthquake occurs, there is an urgent need for EOSs to obtain the photographs of affected areas in a real-time manner. Generally, the photographs are expected to be acquired within a few hours or even minutes for conducting damage assessment and planning rescue policies in a timely manner. Missing deadlines on getting photographs may greatly affect damage assessment and rescue resulting in catastrophic consequences [3].

Apparently, the aforementioned real-time application must then incorporate inherent high reliability features. Hence, the management and control system of multiple EOSs on the ground must ensure its functional and timing correctness even when faults occur on some EOSs. Consequently, providing fault-tolerant mechanisms for such a system with multiple EOSs is mandatory owing to the critical nature of tasks executing on EOSs. Furthermore, in this real-time applications, the arrival times and count of these earth-observation tasks are uncertain, which greatly intensifies the difficulty of developing effective fault-tolerant mechanisms.

Scheduling has been regarded as an efficient approach to helping yield high performance in satellite imaging. The most obvious difference between satellite scheduling and other scheduling problems is that tasks can only be executed in visible time windows of EOSs [4]. Bianchessi *et al.* [1] investigated an improved Tabu search algorithm to solve the multi-satellite, multi-orbit, and multi-user scheduling problem, and evaluated the quality of the solution with an upper bounding procedure based on column generation algorithm. Frank *et al.* [5] modeled this problem using a constraint-based language, and adopted a stochastic greedy search algorithm on the basis of a heuristic guidance strategy to find solutions. Soma *et al* [6]. constructed a mathematical model for multi-satellite scheduling, and developed an optimization approach based on a genetic algorithm. Zufferey *et al.* [7] employed the best ingredients of the graph coloring techniques to the satellite range scheduling problems, proposing two new algorithms: a Tabu search and an adaptive memory algorithm. Globus *et al*. [2] analyzed and concluded the problems of satellite scheduling, proposing the evolutionary algorithm to solve the multi-satellite scheduling problems. It should be noted that the aforementioned static scheduling schemes lack the immediate and dynamic response, not being fit for the applications with uncertain arrival times.

In recent years, more and more attention has been drawn towards the dynamic (on-line) scheduling. For example, Wang *et al.* [8] investigated an approach to solving the dynamic scheduling problem focused on distributed imaging satellites. Billups *et al.* [9] addressed both static and dynamic scheduling issues and studied several solutions including a greedy dynamic method, genetic algorithms, integer programming based approaches and a graph theory approach. Liao and Yang [10] put forward a nominal planning with rolling horizon and lagrangian relaxation technique and suggested a dynamic policy FAHA to adjust the nominal planning with updated

0190-3918/14 $31.00 © 2014 IEEE
DOI 10.1109/ICPP.2014.58

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

information. However, the above dynamic scheduling schemes were not designed for real-time applications, and thus cannot guarantee the deadlines of real-time tasks.

Traditionally, many EOSs are not permanently in contact with a ground control station. However, it is feasible to transfer command and detect the state quickly by relay satellites, especially in emergency. On this basis, we recently proposed some dynamic scheduling algorithms with task merging techniques for real-time applications [11], [12], which greatly improve the effectiveness of scheduling on EOSs in emergency.

Although many satellite scheduling algorithms have been proposed, to the best of our knowledge, no work focuses on satellite fault-tolerant study untill now. In some relevant fields, e.g., clusters, grids, and other multiple processor systems, fault-tolerant scheduling has been extensively investigated (e.g., [13], [14], [15], [16]). However, these policies are not suitable for fault-tolerant scheduling on EOSs due to the unique features of EOSs. To bridge the gap, we attempt to design and implement novel fault-tolerant scheduling algorithms for real-time tasks on multiple EOSs. Specifically, our approach focuses on aperiodic, independent, real-time tasks and employs the Primary-Backup model (PB, in short) in which primary-backup overlapping, backup-backup overlapping and task merging techniques are considered within the time window constraints.

II. Problem Depictions

A. Task Model

We consider a set $T = \{t_1, t_2, ..., t_n\}$ of independent real-time tasks that arrive aperiodically with non-preemptive execution. Because the Primary-Backup (PB) model is employed in our fault-tolerant scheduling scheme, each task t_i has two copies, i.e., its primary copy t_i^P and backup copy t_i^B that execute on different satellites. For a given task $t_i \in T$, it can be modeled by $t_i = (a_i, d_i, rs_i)$, where a_i, d_i, and rs_i represent t_i's arrival time, deadline, and resolution requirement, respectively. The resources in this paper refer to the sensors on EOSs to take photographs of targets.

Let $R = \{r_1, r_2, ..., r_m\}$ be a satellite resource set. Given a resource $r_j \in R$, it is denoted by $r_j = (du_j, \sigma_j, s_j, b_j, o_j, as_j, msg_j, bor_j)$, where $du_j, \sigma_j, s_j, b_j, o_j, as_j, msg_j$, and bor_j are the duration of task execution, field of view angle, slewing pace, start-up time, retention time of shutdown, attitude stability time, maximum slewing angle, and the best ground observation resolution, respectively [11].

Let f_i^P and f_i^B be the finish times of primary copy t_i^P and backup copy t_i^B, respectively. Let $ST^P = (st_{ij}^P)_{n\times m}$ be a start time matrix of tasks' primary copies, where element st_{ij}^P represents the start time of t_i^P on resource r_j. Likewise, $ST^B = (st_{ij}^B)_{n\times m}$ is a start time matrix of tasks' backup copies, where element st_{ij}^B denotes the start time of t_i^B on resource r_j. Correspondingly, we denote $FT^P = (ft_{ij}^P)_{n\times m}$ and $FT^B = (ft_{ij}^P)_{n\times m}$ as the finish time matrix of primary copies and backup copies. $r(t_i^P)$, and $r(t_i^B)$ represents the different satellite resources to which t_i^P and t_i^B are allocated, respectively.

We denote $AO_{ij}^P = \{ao_{ij1}^P, ao_{ij2}^P, ..., ao_{ijK_{ij}}^P\}$ as a set of available opportunities of t_i^P on resource r_j, and AO_i^P is the available opportunity set of t_i^P on all resources, i.e., $AO_i^P = \bigcup_{r_j \in R} AO_{ij}^P$. For a given available opportunity $ao_{ijk}^P \in AO_i^P$, $ao_{ijk}^P = \{[ws_{ijk}^P, we_{ijk}^P], \theta_{ijk}^P\}$, where $[ws_{ijk}^P, we_{ijk}^P]$ is the time window of observation and θ_{ijk}^P is the slewing angle for primary copy t_i^P. An example of available opportunity ao_{ijk}^P can be seen in Fig. 1.

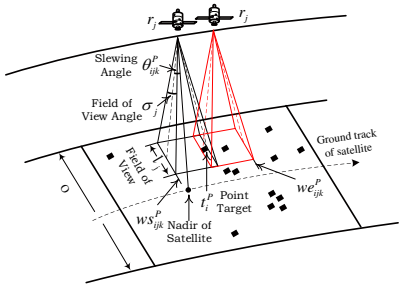


Fig. 1. An Example of Available Opportunity ao_{ijk}^P.

Correspondingly, we have $AO_{ip}^B = \{ao_{ip1}^B, ao_{ip2}^B, ..., ao_{ipQ_{ip}}^B\}$, where $p \neq j$. $AO_i^B = \bigcup_{r_p \in R} AO_{ip}^B$, and $ao_{ipq}^B = \{[ws_{ipq}^B, we_{ipq}^B], \theta_{ipq}^B\}$. ao_{ijk}^P and ao_{ipq}^B are valid available opportunities if 1) $we_{ijk}^P > a_i$, and $we_{ipq}^B > a_i$; 2) $ws_{ijk}^P < d_i$, and $ws_{ipq}^B < d_i$; 3) $\theta_{ijk}^P \in [-msg_j, msg_j]$, and $\theta_{ipq}^B \in [-msg_j, msg_j]$. Within the valid slewing angle, the count of available opportunities cao_i^P of task t_i^P can be derived from (1) as:

$$cao_i^P = K_a + K_b + K_c, \tag{1}$$

where

$$K_a = \begin{cases} 0, & \text{if } we_{ijk}^P \leq a_i \leq we_{ijk+1}^P, \\ 1, & \text{if } a_i < we_{ijk}^P. \end{cases} \tag{2}$$

$$K_b = \#\left\{ao_{ijk}^P | ws_{ijk}^P > a_i, we_{ijk}^P < d_i\right\}, \tag{3}$$

$$K_c = \begin{cases} 0, & \text{if } we_{ijk}^P \leq d_i \leq we_{ijk+1}^P, \\ 1, & \text{if } d_i < we_{ijk}^P. \end{cases} \tag{4}$$

Similarly, the count of cao_i^B can be calculated like cao_i^P.

Decision variable matrix $X^P = (x_{ijk}^P)_{n\times m\times K_{ij}}$ is used to reflect a mapping of primary copies, where element x_{ijk}^P equals "1" if t_i^P has been allocated to the kth available opportunity on resource r_j; otherwise, x_{ijk}^P is "0". Likewise, $X^B = (x_{ipq}^B)_{n\times m\times Q_{ip}}$, in which an element x_{ipq}^B equals "1" if t_i^B has been assigned to the qth available opportunity on resource r_p. Consequently, $r(t_i^P) = j \Leftrightarrow x_{ijk}^P = 1$ and $r(t_i^B) = p \Leftrightarrow x_{ipq}^B = 1$. Z_i^P and Z_i^B represent all possible schedules for primary copy t_i^P and backup copy t_i^B, respectively. $z_i^P \in Z_i^P$ is a scheduling decision of t_i^P. Similarly, $z_i^B \in Z_i^B$ is a scheduling decision of t_i^B. The

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

observation resolution (unit: meter) of t_i^P employing the scheduling decision t_i^P is $or(z_i^P)$ that can be computed from (6) below:

$$or(z_i^P) = \frac{bor_j \cdot L_{ijk}^P}{H_j}, \tag{5}$$

$L_{ijk}^P = (H_j + R)\cos\theta_{ijk}^P - \sqrt{R^2 - (H_j + R)^2 sin^2\theta_{ijk}^P}$, in which R is the earth radius, θ_{ijk}^P is the observation angle of task t_i^P on kth time window on resource r_j and H_j is the orbit height of r_j. $or(z_i^B)$ can use a similar computation like (6). z_i^P and z_i^B are feasible scheduling decisions if 1) $f_i^P \leq d_i$ and $f_i^B \leq d_i$; 2) $or(z_i^P) \leq rs_i$ and $or(z_i^B) \leq rs_i$.

B. Fault Model

The fault model described in this paper includes the following assumptions.

- Faults on satellites are transient or permanent, and independent, namely, a fault on one satellite will not affect another satellite.
- Since the probability that two satellites failing simultaneously is extremely small, we assume that only one satellite fails at one time instant. The backup copies can be successfully finished if their corresponding primary copies are on the failed satellite.
- A fault-detection mechanism is available to detect satellite failures. New tasks will not be allocated to that known failed satellite.

In this study, our primary objective is to accommodate as many tasks and then to minimize the observation resolutions of all accepted tasks under timing constraints.

III. Overlapping Design and Analysis

To efficiently boost the utilization of satellite resources and thereby to improve the schedulability, we attempt to employ the overlapping mechanism including backup-backup (BB, in short) and primary-backup (PB, in short) overlapping in our study based on the consideration that a majority of backup copies only occupy resources but do not execute as the probability of a satellite's fault is tiny.

Property 1. One satellite's fault can be tolerated if and only if the primary copy and the backup copy of a task are allocated to two different resources, i.e.,

$$\forall t_i \in T, r(t_i^P) \neq r(t_i^B). \tag{6}$$

Property 1 indicates that the primary copy and the backup copy of a task cannot be assigned to the same satellite, otherwise both copies cannot be successfully finished when the resource meets a fault, and thus fail to realize fault-tolerance.

Property 2. The start time of backup copy t_i^B is later than the finish time of primary copy t_i^P, i.e.,

$$\forall t_i \in T, st_{ip}^B > ft_{ij}^P, j \neq p. \tag{7}$$

Property 3. Suppose t_i^P finishes successfully within its deadline, the time slot occupied by t_i^B will be reclaimed immediately and t_i^B will not be executed.

Property 4. The overlapping mechanism is employed on condition that the primary copies and backup copies can be executed on available opportunities.

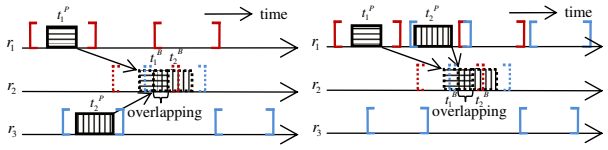
On the basis of Property 4, the observation resolution of each task must be better than or equal to the users' resolution requirements. This comes to the following property:

Property 5. A task can be allocated only if its observation resolution is equal or smaller than that of the user's requirement, i.e.,

$$\forall t_i^P \in T, t_i^B \in T, or(z_i^P) \leq rs_i \text{ and } or(z_i^B) \leq rs_i. \tag{8}$$

A. BB Overlapping

In order to improve the resource utilization, one of our considerations is to employ the backup-backup (i.e., BB) overlapping mechanism in which a backup copy of one task may overlap with backup copies of other tasks. Fig. 2 illustrates two examples of BB overlapping.



(a) Feasible BB Overlapping (b) Unfeasible BB Overlapping

Fig. 2. Examples of BB Overlapping. The red solid pane represents the available opportunities of task t_1^P, the red dashed pane represents the available opportunities of task t_1^B, the blue solid pane represents the available opportunities of task t_2^P, and the blue dashed pane represents the available opportunities of task t_2^B.

We can observe from Fig. 2(a) that t_1^B and t_2^B are able to overlap. The reason is derived from the following four-fold analysis.

- If resources r_1, r_2 and r_3 do not occur a fault, t_1^P and t_2^P will be finished successfully, and thus t_1^B and t_2^B do not need to be executed any more. Therefore, t_1^B and t_2^B can overlap.
- If resource r_1 meets a fault, t_1^P will then fail, so t_1^B has to be executed. According to the assumption in Section IIB, r_2 and r_3 will not fail, thus t_2^P can be successfully finished. Since $st_{22}^B > ft_{23}^P$, after the completion of t_2^P, the time slot occupied by t_2^B will be reclaimed, which will not result in a conflict between t_1^B and t_2^B with respect to execution time. Hence, t_1^B and t_2^B can overlap.
- If a fault occurs at resource r_3, t_2^P cannot be successfully finished, and there is no timing conflict between t_1^B and t_2^B with the explanation as above analysis.
- If resource r_2 encounters a fault, t_1^B and t_2^B cannot be finished. Based on our assumption, r_1 and r_2 can work normally, then t_1^P and t_2^P can be finished successfully, so the overlapping of t_1^B and t_2^B will not affect fault-tolerance.

It should be noted that t_1^P and t_2^P cannot be allocated to the same resource. An unfeasible allocation example can be seen in Fig. 2(b).

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

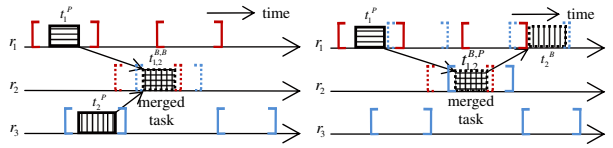
From Fig. 2(b), we can easily find that if resource r_1 encounters a fault, t_1^B and t_2^B must be executed. However, due to the overlapping of t_1^B and t_2^B, there exists a timing conflict between t_1^B and t_2^B. Consequently, we get the following property.

Property 6. The primary copies $t_i^P, t_j^P, \cdots, t_k^P$ whose corresponding backup copies $t_i^B, t_j^B, \cdots, t_k^B$ overlap cannot be allocated to the same resource, i.e.,

$$r(t_i^P) \cap r(t_j^P) \cap \cdots \cap r(t_k^P) = \emptyset. \tag{9}$$

According to the features of satellite observation, i.e., a scene imaged by a satellite sensor is with a certain size, some adjacent targets in the same scene can be observed at the same time. This unique characteristic motivates us to merge some tasks while overlapping to further improve the schedulability.

Now, we give an example to analyze this condition. Fig. 3(a) plots the example.



(a) BB Merging (b) PB Merging

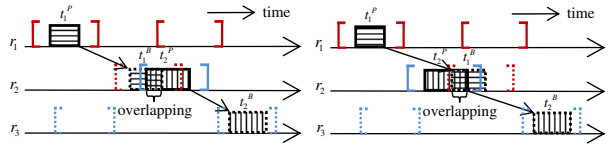
Fig. 3. Examples of Merging.

In Fig. 3(a), $t_{1,2}^{B,B}$ represents the merged task by t_1^B and t_2^B. We can find that a fault occurring in any resource, r_1, r_2, or r_3 will not affect fault-tolerance, i.e., t_1 and t_2 can be successfully finished.

Additionally, if some backup copies can be merged, their corresponding primary copies are capable of being allocated to the same resources.

B. PB Overlapping

Another consideration of the overlapping mechanism in our study is primary-backup (i.e., PB) overlapping in which the primary copy of a task can overlap with backup copies of other tasks. An example of PB overlapping is shown in Fig. 4(a).



(a) Feasible PB Overlapping (b) Unfeasible PB Overlapping

Fig. 4. Examples of PB Overlapping.

In Fig. 4(a), t_1^B can overlap with t_2^P, which can be analyzed from the four cases as follows:

- If resources r_1, r_2 and r_3 work normally, t_1^B and t_2^B do not need to be executed after t_1^P and t_2^P finish successfully. Then, the occupied time slots will be reclaimed. t_1^B and t_2^P have no chance to execute simultaneously. So t_1^B and t_2^P can overlap in this case.
- If resource r_1 occurs a fault, t_1^B has to be executed. Based on our assumption, r_2 and r_3 will work normally. Since t_1^B and t_2^P are overlapping in terms of timing, to make t_1^B finish successfully, t_2^B must be chosen to execute, giving up the execution of t_2^P. By this operation, it is feasible for t_1^B and t_2^P to overlap.
- If resource r_2 fails, t_1^P and t_2^B can be successfully finished on the basis of our assumption, so t_1^B and t_2^P are able to overlap each other.
- If resource r_3 encounters a fault, t_1^P and t_2^P can be finished. Because t_1^B will be reclaimed after the completion of t_1^P, there is no timing conflict in t_1^B and t_2^P. Hence, the overlapping of t_1^B and t_2^P is reasonable.

However, if the start time of t_j^P is earlier than that of t_i^B, t_j^P cannot overlap with t_i^B. An example of unfeasible PB overlapping is depicted in Fig. 4(b) as below:

Suppose that resource r_1 encounters a fault, t_1^B must execute. But at this time t_2^P is executing and cannot be interrupted, so there exists a time conflict between t_1^B and t_2^P. Property 7 describes a constraint if a primary copy can overlap with a backup copy.

Property 7. If a primary copy t_i^P can overlap with a backup copy t_j^B on resource r_p, the start time of t_i^P must be later than that of t_j^B, i.e.,

$$st_{ip}^P > st_{jp}^B. \tag{10}$$

Similar to BB overlapping, t_1^P and t_2^B cannot be allocated to the same resource. We can conclude the following property.

Property 8. If a primary copy t_i^P and backup copies $t_j^B, \cdots, t_k^B$ overlap, t_i^B and $t_j^P, \cdots, t_k^P$ cannot be allocated to the same resource, that is,

$$r(t_i^B) \cap r(t_j^P) \cap \cdots \cap r(t_k^P) = \emptyset. \tag{11}$$

However, based on the example in Fig. 4(a), if t_1^B and t_2^P can be merged, t_1^P and t_2^B are able to be allocated to the same resource as shown in Fig. 3(b).

It is clear that t_1^P and t_2^B can be allocated to the same resource because t_1 and t_2 can be finished successfully no matter if r_1 or r_2 encounters a fault. The merged task $t_{1,2}^{B,P}$ comes from t_1^B and t_2^P.

C. PP Merging

From the analysis of the above overlapping technologies, it can be found that primary copies cannot be overlapped because this kind of overlapping fails to solve the fault-tolerance issue. However, multiple primary copies can be merged while conducting fault-tolerant scheduling besides the PB merging and BB merging discussed in PB overlapping and BB overlapping. We present three successful examples of PP merging in Fig. 5.

Fig. 5(a) illustrates that if t_1^P merges with t_2^P, one choice is that their corresponding backup copies t_1^B and t_2^B are allocated to two other different resources. Following this operation, we

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

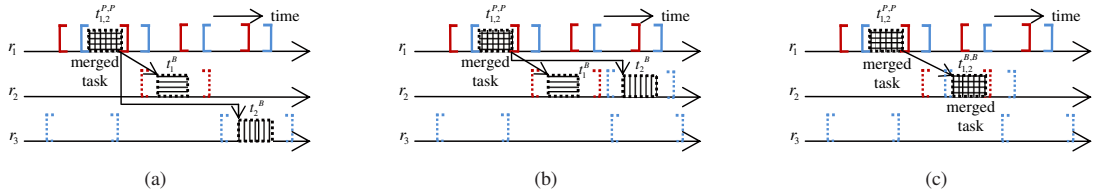


Fig. 5. Examples of Successful PP Merging.

can easily get that no matter which resource occurs a fault, t_1 and t_2 can be successfully finished based on our assumption.

From Fig. 5(b), we can find that if resource r_1 fails, t_1^B and t_2^B are able to finish successfully, and if resource r_2 encounters a fault, $t_{1,2}^{P,P}$ can successfully finish on the basis of the aforementioned assumption. Thus, we can conclude that if t_1^P merges with t_2^P, it is rational that t_1^B and t_2^B are allocated to the same resource except $r(t_{1,2}^{P,P})$ and at the same time t_1^B and t_2^B have no overlapping.

As for Fig. 5(c), it is clear that if t_1^P merges with t_2^P, it is also feasible to make t_1^B merge with t_2^B, and note $t_{1,2}^{P,P}$ and $t_{1,2}^{B,B}$ need to be allocated to a different resource.

D. Merging Constraints

Let t_a^X and t_b^Y represent two tasks ready to be merged, where X and Y denote any kind of copy, i.e., X and Y may be P or B.

First, t_a^X and t_b^Y must have an overlapped time window, namely,

$$\begin{aligned}\forall t_a^X \in T, t_b^Y \in T, r_j \in R, ao_{ajk}^X \in AO_{aj}^X, ao_{bjq}^Y \in AO_{bj}^Y,\\ we_{ajk}^X > ws_{bjq}^Y, \text{ if } we_{bjq}^Y > we_{ajk}^X,\\ we_{bjq}^Y > ws_{ajk}^X, \text{ if } we_{ajk}^X > we_{bjq}^Y.\end{aligned} \tag{12}$$

Second, the overlapped time window must be equal to or larger than the execution time of t_a^X and t_b^Y, i.e.,

$$\min\{we_{ajk}^X, we_{bjq}^Y\} - \max\{ws_{ajk}^X, ws_{bjq}^Y\} \geq du_j \ . \tag{13}$$

Third, the scene of resource r_j must be able to cover the targets of both t_a^X and t_b^Y by adjusting its observation angle, that is,

$$|\theta_{ajk}^X - \theta_{bjq}^Y| \leq \sigma_j \ . \tag{14}$$

Fourth, the observation resolution of a merged task t_{ab}^{XY} must be equal to or better than the users' requirements in terms of observation resolutions, which comes to the following constraint:

$$or(z_{ab}^{XY}) \leq rs_a, \text{ and } or(z_{ab}^{XY}) \leq rs_b \ . \tag{15}$$

If t_a^X merges with t_b^Y, the time window of merged task t_{ab}^{XY} becomes:

$$tw_{ab,j,kq}^{XY} = |\max\{ws_{ajk}^X, ws_{bjq}^Y\}, \min\{we_{ajk}^X, we_{bjq}^Y\}|. \tag{16}$$

Also, the new slewing angle is set to be:

$$\theta_{ab,j,kq}^{XY} = (\theta_{a,j,k}^X + \theta_{b,j,q}^X)/2. \tag{17}$$

So, the available opportunity of t_{ab}^{XY} is

$$ao_{ab,j,kq}^{XY} = \{tw_{ab,j,kq}^{XY}, \theta_{ab,j,kq}^{XY}\}. \tag{18}$$

Apparently, the count of the available opportunities of the merged task t_{ab}^{XY} equals the count of the merging opportunities between t_a^X and t_b^Y. Hence, the merged task t_{ab}^{XY} may also have multiple available opportunities, which provides multiple choices to optimize our scheduling objectives.

IV. Fault-Tolerant Scheduling Algorithm FTSS

A. Primary Copies Scheduling

In order to make a primary copy and its corresponding backup copy be allocated successfully, the primary copy should be executed as early as possible to save more time slots for the backup copy to be allocated before its deadline.

To get the earliest start time of a primary copy, the preparation time and ready time are needed.

Definition 1. Preparation time $p_{i-1,i,j}^{XP}$ is the time required to start task t_i^P from its previous task t_{i-1}^X due to the change of slewing angle, i.e.,

$$p_{i-1,i,j}^{XP} = o_j + \frac{|\theta_{i-1,j,k}^X - \theta_{i,j,q}^P|}{s_j} + as_j + b_j. \tag{19}$$

Definition 2. Ready time r_{ij}^P is the time from which t_i^P can be started,

$$r_{ij}^P = \max\{p_{i-1,i,j}^{XP} + ft_{i-1,j}, a_i\}. \tag{20}$$

Property 9. A primary copy t_i^P can be executed in an available opportunity ao_{ijk}^P, which could be operated by the following three ways:

- t_i^P is able to merge with a task (or a merged task);
- t_i^P overlaps with a task (or a merged task);
- ao_{ijk}^P has an idle time slot sufficiently being enough to accommodate t_i^P.

Either way, the finish time of t_i^P must be earlier than the deadline of t_i, i.e.,

$$f_i^P < d_i. \tag{21}$$

In the first way, constraints (12)−(15) must be satisfied to guarantee t_i^P is able to merge with a task (or a merged task). The est_{ijk}^P is equal to the start time of task (or merged task) t_a^X needed to merge with, i.e.,

$$est_{ijk}^P = st_{aj}^X. \tag{22}$$

For the second way, t_i^P can overlap with a backup copy or overlap with a merged task that cannot derive from any

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

primary copies, i.e., if t_i^P can overlap with a merged task $t_{i_1,i_2,\cdots,i_k}^{X_1,X_2,\cdots,X_k}$,

$$X_1 \neq P \text{ and } X_2 \neq P \text{ and } \cdots \text{ and } X_k \neq P. \quad (23)$$

Since t_i^P can only overlap with backup copies or merged tasks containing only backup copies, it is necessary to search the first task t_a^X (maybe a backup copy or a merged task constituted only by backup copies), then the est_{ijk}^P can be obtained by:

$$est_{ijk}^P = \max\{r_{ij}^P, st_{aj}^X + \varepsilon\}, \quad (24)$$

where ε is the time to cancel the execution of t_i^P if t_a^X has to be executed.

With regard to the third way, the earliest start time est_{ijk}^P also needs to be calculated. Without loss of generality, before est_{ijk}^P can be computed, it is assumed that $t_{i_1}, t_{i_2}, \cdots, t_{i_q}$ have been allocated to the kth available opportunity on resource r_j. To simplify the symbol, they can be either primary copies or backup copies. Also, they can be single tasks or merged tasks. The idle time slots on ao_{ijk}^P are $[ws_{ijk}^P, st_{i_1}], [ft_{i_1}, st_{i_2}], \cdots, [ft_{i_{q-1}}, st_{i_q}], [ft_{i_q}, we_{ijk}^P]$. To obtain the est_{ijk}^P, all these time slots should be scanned by timing in ascending order. As a result, the first idle time slot $[ft_{i_k}, st_{i_{k+1}}]$ that satisfies the following inequality is selected:

$$\max\{r_{ij}^P, ft_{i_k} + p_{i_k,i,j}^{XP}\} + du_j \leq st_{i_{k+1}}. \quad (25)$$

Hence, the est_{ijk}^P can be determined by:

$$est_{ijk}^P = \max\{r_{ij}^P, ft_{i_k} + p_{i_k,i,j}^{XP}\}. \quad (26)$$

In our FTSS, we let the start time of a primary copy be its earliest start time so as to execute it as early as possible.

The pseudocode of primary copy scheduling in FTSS is outlined in Algorithm 1.

B. Backup Copies Scheduling

Concerning the backup copy whose primary copy has been allocated, it should be executed as late as possible to save earlier time slots guaranteeing more later-arriving tasks finishing before their deadlines. Additionally, the idle time slot that is smaller should be selected to make room for other tasks to enhance schedulability.

Property 10. A backup copy t_i^B can be allocated in an available opportunity ao_{ipq}^B, which could be operated by the following three ways:

- t_i^B is able to merge with a task (or a merged task);
- t_i^B overlaps with a task (or a merged task);
- ao_{ipq}^B has an idle time slot sufficiently to accommodate t_i^B.

In the first way, constraints (12)−(15) must be satisfied to guarantee t_i^B being able to merge with a task (or a merged task). The lst_{ipq}^B is equal to the start time of task (or merged task) t_b^X needed to merge with, i.e.,

$$lst_{ipq}^B = st_{bp}^X. \quad (27)$$

Algorithm 1 Primary Copies Scheduling in FTSS

```
for each new task t_i, schedule its primary copy t_i^P do
    allocationTag ← FALSE; mergeTag ← FALSE;
    for each resource r_j in the system do
        Calculate t_i^P's available opportunities AO_ij^P;
        Calculate t_i^P's valid available opportunities by (5);
        PMerging();
        if allocationTag ≠ TRUE then
            POverlapping();
            PInserting();
        end if
        if allocationTag == TRUE and mergeTag == FALSE then
            est_ij^P ← min{est_ij^P', est_ij^P''};
        end if
        if allocationTag == TRUE then
            Put allocation scheme of t_i^P into set aScheme_i^P;
        else
            Reject t_i^P and t_i^B;
        end if
    end for
    Sort the allocation schemes in aScheme_i^P by t_i^P's start time in ascending order;
    aScheme_i^P(u) ← the former u% schemes in aScheme_i^P;
end for
```

Algorithm 2 Function PMerging()

```
for each valid available opportunity ao_ijk^P do
    if t_i^P can merge with a task (or a merged task) t_a^X allocated in tw_ijk^P then
        Merge t_i^P with t_a^X;
        est_ijk^P ← st_aj^X;
        Calculate t_i^P's finish time f_i^P;
        if f_i^P < d_i then
            mergeTag ← TRUE; allocationTag ← TRUE;
            break;
        end if
    end if
end for
```

Now, we discuss the latest start time of a task's backup copy in the second and third ways. For any backup copy t_i^B, the latest start time lst_{ijk}^B can be obtained by scanning the time windows from right to left.

$$lst_{ipq}^B = \begin{cases} \min\{we_{ipq}^B, d_i\} - du_j, & \text{if } lts = \text{idle} \vee ol(t_i^B), \\ st_{bp}^X - p_{ibp}^{BX} - du_j, & \text{if } ts = \text{idle} \vee ol(t_i^B). \end{cases} \quad (28)$$

where lts denotes the latest time slot and $ol(t_i^B)$ denotes that t_i^B can overlap with other tasks (or merged tasks) in the latest time slot. st_{bp}^X denotes the start time of task t_b^X that executes following t_i^B and cannot overlap with it. ts represents the time slot before that occupied by t_b^X. p_{ibp}^{BX} is the preparation time required to start task t_b^X after finishing t_i^B.

The start time of a backup copy is set to the latest start time in our FTSS. The pseudocode of backup copy scheduling in FTSS is outlined in Algorithm 5.

V. Performance Evaluation

We evaluate in this section the performance of the proposed FTSS algorithm. To demonstrate the performance improve-

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

TABLE I
PARAMETERS OF SATELLITE SENSORS

Sensor	Satellite	bs(m)	msg(deg)	FOV	du^*(s)	s^*(deg/s)	b^*(s)	o^*(s)	as^*(s)	Height(km)
Sensor1	IKONOS-2	4	45	0.931	1	0.5	5	5	10	681
Sensor2	QuickBird-2	2.44	25	2.1	1	0.5	5	5	10	450
Sensor3	SPOT-4	10	27	4.2	1	0.5	5	5	10	832
Sensor4	SPOT-5	10	27	2.09	1	0.5	5	5	10	822
Sensor5	ORBVIEW-03	1	27	1.0	1	0.5	5	5	10	470
Sensor6	JB-3A	3	32	5.72	1	0.5	5	5	10	300
Sensor7	EROS-A01	1.9	45	1.6	1	0.5	5	5	10	500
Sensor8	CBERS-1	19.5	32	8.32	1	0.5	5	5	10	778
Sensor9	CBERS-2	10	32	8.32	1	0.5	5	5	10	778
Sensor10	ERS-2	10	25	7.34	1	0.5	5	5	10	780

Algorithm 3 Function POverlapping()

for each valid available opportunity ao_{ijk}^P **do**
  if t_i^P can overlap with a task (or a merged task) t_a^X allocated in tw_{ijk}^P based on Properties (7) and (8) **then**
    Calculate t_i^P's earliest start time est_{ijk}^P by (29);
    Calculate t_i^P's finish time f_i^P;
    if $f_i^P < d_i$ **then**
      Calculate t_i^P's observation resolution $or(z_i^P)$ by (6);
      if $or(z_i^P) \leq rs_i$ **then**
        $allocationTag \leftarrow$ TRUE; $est_{ij}^{P'} \leftarrow est_{ijk}^P$;
        break;
      end if
    end if
  end if
end for

Algorithm 4 Function PInserting()

for each valid available opportunity ao_{ijk}^P **do**
  if ao_{ijk}^P has an idle time slot to accommodate t_i^P **then**
    Calculate t_i^P's earliest start time est_{ijk}^P;
    Calculate t_i^P's finish time f_i^P;
    if $f_i^P < d_i$ **then**
      Calculate t_i^P's observation resolution $or(z_i^P)$ by (6);
      if $or(z_i^P) \leq rs_i$ **then**
        $allocationTag \leftarrow$ TRUE; $est_{ij}^{P''} \leftarrow est_{ijk}^P$;
        break;
      end if
    end if
  end if
end for

Algorithm 5 Backup Copies Scheduling in FTSS

for each new task t_i whose primary copy t_i^P has not been rejected, schedule its backup copy t_i^B **do**
  $allocationTag \leftarrow$ FALSE; $value \leftarrow +\infty$;
  for each allocation scheme of t_i^P in $aScheme_i^P(u)$ **do**
    $allocationTag' \leftarrow$ FALSE; $mlst \leftarrow 0$; $nmlst \leftarrow 0$;
    for each resource r_j except $r(t_i^P)$ in the system **do**
      Calculate t_i^B's available opportunities AO_{ip}^B;
      Calculate t_i^B's valid available opportunities as (5);
      BMerging();
      if $allocationTag \neq$ TRUE and $mlst == 0$ **then**
        BOverlappingOrInserting();
      end if
    end for
    if $allocationTag' ==$ TRUE and $mlst \neq 0$ **then**
      Put allocation scheme with maximal $mlst$ of t_i^B into set $aScheme_i^B$;
    else if $allocationTag' ==$ TRUE and $mlst == 0$ **then**
      Put allocation scheme with maximal $nmlst$ of t_i^B into set $aScheme_i^B$;
    end if
  end for
  if $allocationTag ==$ FALSE **then**
    Reject t_i^P and t_i^B;
  end if
  for each primary copy allocation scheme in $aScheme_i^P(u)$ and the corresponding backup copy allocation scheme in $aScheme_i^B$ **do**
    Calculate $value_i = \frac{st_i^P}{st_i^B}/(or(z_i^P) + or(z_i^B))$;
    if $value_i < value$ **then**
      Select the current task allocation scheme;
    end if
  end for
end for

ments gained by FTSS, we quantitatively compare it with three baseline algorithms: a fault-tolerant satellite scheduling algorithm without merging (NMFTSS); a fault-tolerant satellite scheduling algorithm without overlapping (NOFTSS); a fault-tolerant satellite scheduling algorithm without merging and overlapping (NMNOFTSS).

- NMFTSS: Different from FTSS, NMFTSS does not take task merging into account. From the performance comparisons between NMFTSS and FTSS, we can evaluate the effectiveness of task merging.
- NOFTSS: As a variant of FTSS, task overlapping is not considered in NOFTSS. The performance discrepancy between NOFTSS and FTSS is able to validate the value of overlapping.
- NMNOFTSS: NMNOFTSS considers task insertion in the waiting sequence only, neither task merging nor overlapping is considered.

The performance metrics by which we evaluate the system performance include the following:

- Guarantee Ratio (GR) defined as: GR=Total number of tasks guaranteed to meet their deadlines/Total number of tasks $\times$ 100%;

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

Algorithm 6 Function BMerging()

```
for each valid available opportunity ao^B_ipq do
    if t^B_i can merge with a task (or a merged task) t^X_b allocated in tw^B_ipq then
        Merge t^B_i with t^X_b;
        lst^B_ipq ← st^X_bp;
        if lst^B_ipq > nmlst then
            mlst ← lst^B_ipq;
        end if
        allocationTag ← TRUE; allocationTag' ← TRUE;
        break;
    end if
end for
```

Algorithm 7 Function BOverlappingOrInserting()

```
for each valid available opportunity ao^B_ipq do
    if t^B_i can be accommodated by overlapping or inserting then
        Calculate t^B_i's latest start time lst^B_ipq by (33);
        Calculate t^B_i's observation resolution or(z^B_i) as (6);
        if or(z^B_i) ≤ rs_i then
            allocationTag ← TRUE; allocationTag' ← TRUE;
            if lst^B_ipq > nmlst then
                nmlst ← lst^B_ipq;
            end if
            break;
        end if
    end if
end for
```

- Observation Resolution (OR) used to test the average observation resolution of accepted tasks.

A. Simulation Method and Parameters

To validate the performance improvements of FTSS, the targets are randomly generated in the area: latitude $-30°\sim60°$ and longitude $0\sim150°$. The number of targets are 200, 400, 600, 800, 1000, 1200, respectively. According to extensive literature such as [17], [18], ten sensors on different satellites are considered in this paper. The parameters of the sensors are presented in Table 1, and the orbit models of satellites are obtained from the Satellite Tool Kit (STK), where the parameters with (*) denote the designed values.

- The arrival time of a task is denoted as $a_i = a_{i-1} + intervalTime$, where $intervalTime$ is a random positive real number, uniformly distributed in $[a, b]$, $a_0 = 0$;
- The deadline d_i of task t_i is described as $d_i = a_i + Deadline$, where $Deadline \sim N(baseDeadline, baseDeadline/10)$.

Table 2 gives the simulation parameters and their values. The STK presents the running status of earth-observation system composed by 10 satellites listed in Table 1. Besides, the random synthetic tasks generated by our simulation parameters are shown on the Earth (Target i denotes the ith task). Fig. 6 shows the 3D graphic when 100 tasks are generated.

B. Performance Impact of Task Number

In this experiment, we investigate the performance impact of the number of tasks that increases from 200 to 1200 with steps of 200. Fig. 7 illustrates the performances of FTSS, NMFTSS, NOFTSS and NMNOFTSS in terms of guarantee ratio and observation resolution.

TABLE II
PARAMETERS FOR SIMULATION STUDIES

Parameter	Value(Fixed)-(Varied)
Task Number	(1000)-(200, 400, 600, 800, 1000, 1200)
Resource Number	(10)
$intervalTime$(h)	([20,40])-([0,20], [20, 40], [40, 60], [60, 80], [80, 100])
u	(50)
$baseDeadline$(h)	(30)-(20, 30, 40, 50, 60, 70)

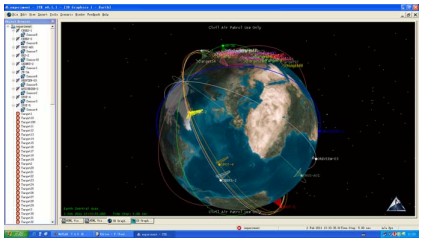
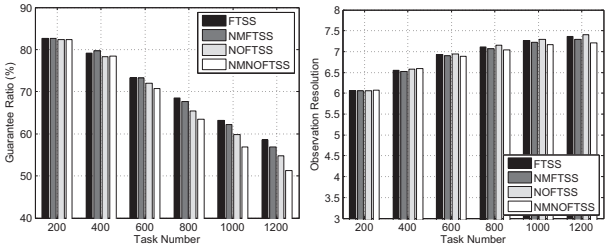


Fig. 6. STK 3D Graphics.



(a) (b)

Fig. 7. Performance Impact of the Number of Tasks.

Fig. 7(a) shows that with the increase of the number of tasks, the guarantee ratios of all algorithms get decreased. This is because in the condition of limited resources, the increase of the number of tasks makes the system workload heavy. Besides, Fig. 7(a) shows that the guarantee ratio of NMNOFTSS is the lowest because neither the overlapping technique nor the task merging strategies are employed in NMNOFTS, which results in a poor resource utilization. On the contrary, FTSS performs best in terms of guarantee ratio. We attribute the result to the adoption of several overlapping techniques in FTSS. In addition, the guarantee ratio of NMFTSS is higher than that of NOFTSS. This is because the constraints of task merging are stricter, which makes the guarantee ratio of NOFTSS a bit lower.

Fig. 7(b) demonstrates that the observation resolutions of the four algorithms increase with the increase of the number of tasks. This is because when the resources are limited, the increase of the number of tasks results in a heavy system

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

workload, which makes the system have to sacrifice observation resolutions for accepting more tasks. It can be found from Fig. 7(b) that when the system workload is light (the task number is less than 400), the observation resolution of FTSS is better than that of NMNOFTSS, which indicates that FTSS can optimize the observation resolution as well as accept more tasks. When the system workload is heavy (the task number is greater than 600), the observation resolution is worse than that of NMNOFTSS. This is due to the fact that FTSS prefers to accept more tasks, and thus the observation resolution degrades inevitably.

C. Performance Impact of Arrival Rate

A group of experiments are carried out in this section to observe the impact of the arrival rate on the algorithms. Parameter *intervalTime* is uniformly distributed among [0,20], [20,40], [40,60], [40,60], [60,80] and [80,100]. The experimental results are depicted in Fig. 8.

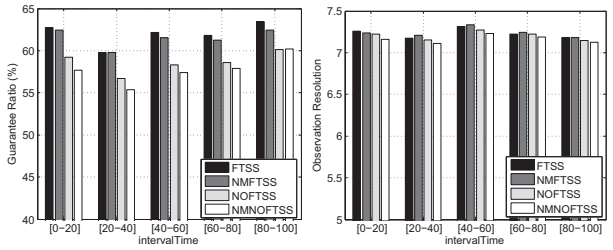


Fig. 8. Performance Impact of Arrival Rate.

Fig. 8(a) demonstrates that when the *intervalTime* increases from [0,20] to [20,40], the guarantee ratios of all the algorithms decrease. The reason is twofold. First, when *intervalTime* is in [20,40], the system workload is still heavy. When new tasks arrive, most of the previous tasks have not been finished. However, the decrease of the arrival rate makes the system tend to accept more tasks which leads to the rejection of the following tasks because the previously accepted tasks have not been finished. Hence, the guarantee ratios decrease a bit. Second, when tasks arrive rapidly, more time windows of tasks can overlap with others, which produces more opportunities for task overlapping and merging. However, when *intervalTime* becomes a little greater (the system workload is still heavy), the opportunities for task overlapping and merging become less, and thus all the guarantee ratios descend. In the period of *intervalTime* increasing from [20,40] to [80,100], the guarantee ratios of all the algorithms increase. This is because the system workload becomes light, and the previous tasks can be finished when new tasks arrive, and thus more tasks can be finished. It is worth noting that the guarantee ratio of FTSS turns out to be higher than those of other algorithms when *intervalTime* varies. We attribute this result to the fact that FTSS employs task overlapping and merging techniques, which improves the resource utilization and scales the number of tasks down. As a result, FTSS exhibits the best performance in terms of guarantee ratio.

From Fig. 8(b), it is found that the differences among the four algorithms are tiny. The observation resolutions become better with the increase of *intervalTime*. The reason is that the system that is in light workload can offer better observation resolutions as well as accept more tasks. Additionally, among the four algorithms, NMNOFTSS has the best observation resolutions while the worst guarantee ratio (see Fig. 8(a)). In emergent situations, the system should accept as many tasks as possible within timing constraints, and meanwhile satisfy users' observation resolutions requirements. Although NMNOFTSS has the smallest observation resolution, it makes the system lose many opportunities to execute tasks. On the contrary, as the employment of task overlapping and task merging techniques, FTSS can improve the guarantee ratio without huge impact on the observation resolutions. The difference between FTSS and NMNOFTSS is only 0.065867. This result proves that FTSS also exhibits a good performance in terms of observation resolution.

D. Performance Impact of Task Deadline

The objective of this experiment is to investigate the impact of the tasks' deadline on the performance of the four algorithms. The parameter *baseDeadline* varies from 20 to 70 with an increment of 10. Fig. 9 illustrates the experimental results.

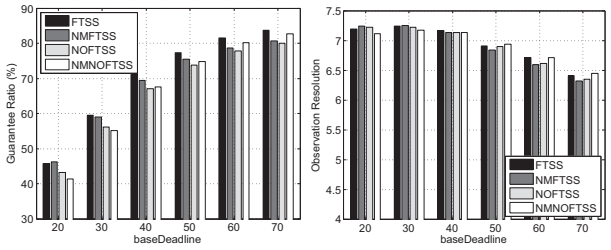


Fig. 9. Performance Impact of Task Deadline.

Fig. 9(a) demonstrates that the guarantee ratios of the four algorithms show ascending trends with the increase of *baseDeadline*. The reason is that the deadline of tasks become loose, which increases the available opportunities for tasks, and consequently reduces the execution conflicts between different tasks. So the guarantee ratios increase. Further, when *baseDeadline* is 20, the guarantee ratio of NMFTSS is a bit higher than that of FTSS. This is because FTSS tends to merge tasks preferentially. However, in the situation of tight deadlines, this characteristic of FTSS causes some tasks to be unable to overlap with others. Thus, the number of accepted tasks decreases. When *baseDeadline* varies from 30 to 70, the guarantee ratio of FTSS is always higher than that of NMFTSS. This result can be attributed to the fact that despite the aforementioned characteristic of FTSS, the opportunities for task overlapping also increase in the condition that the

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.

deadline becomes loose. It should be noted that when the value of *baseDeadline* is between 50 and 70, NONOFTSS always outperforms NMFTSS and NOFTSS, however being inferior to FTSS.

Fig. 9(b) depicts that the observation resolutions of all the algorithms increase a bit at first, and then shows descending trends with the increase of *baseDeadline*. When the deadline becomes a bit looser, the system is still overloaded. It has to sacrifice some tasks' observation resolutions for accepting more tasks. However, when the deadline becomes further loosened, the system is able to accept more tasks and offer good observation resolutions at the same time. The observation resolution of FTSS is a bit higher than that of NMNOFTSS (the average difference is only 0.014623), but FTSS accepts more tasks than NMNOFTSS, and so is more suitable for scheduling tasks in emergent situations.

VI. Conclusions and Future Work

This paper studies the problem of fault-tolerant scheduling for real-time tasks on multiple observation satellites. The fault-tolerant capability is obtained by using primary-backup policy. The scheduling objective is to improve real-time tasks' schedulability and optimizing tasks' observation resolutions within the fault-tolerant requirements. To achieve this objective, this paper first builds a scheduling model based on primary-backup policy, and then analyzes the task overlapping conditions on multiple EOSs and the constraints of task merging elaborately. On this basis, we propose a novel fault-tolerant satellite scheduling algorithm called FTSS, which comprehensively considers task overlapping, task merging and task inserting. FTSS can enhance the resource utilization efficiently, and thus improve the system's performance in terms of guarantee ratios and observation resolutions.

The FTSS algorithm is the first of its kind reported in the literature; it comprehensively addresses the issue of fault-tolerance, real-time, schedulability, and adaptivity. To evaluate the performance of FTSS, we conduct extensive experiments to compare it with three baseline algorithms. The experimental results indicate that FTSS efficiently improves the scheduling quality of others.

The following issues will be addressed in our future studies: Firstly, we will extend our fault-tolerant scheduling model to tolerate multiple satellites' failures and propose corresponding scheduling algorithms. Secondly, we will take the communication time and task dispatching time into consideration to enhance the accuracy of scheduling. Thirdly, we plan to implement FTSS in a real satellite system to test its performance.

References

[1] N. Bianchessi, J. F. Cordeau, J. Desrosiers, G. Laporte, and V. Raymond, "A Heuristic for the Multi-Satellite, Multi-Orbit and Multi-User Management of Earth Observation Satellites," *European J. Operational Research*, vol. 177, no. 2, pp. 750-762, 2007.

[2] A. Globus, J. Crawford, J. Lohn, and A. Pryor, "A Comparison of Techniques for Scheduling Fleets of Earth-Observing," *J. Operational Research Society*, vol. 56, no.8, pp. 962-968, 2003.

[3] K. J. Zhu, J. F. Li, H. X. Baoyin, "Satellite Scheduling Considering Maximum Observation Coverage Time and Minimum Orbital Transfer Fuel Cost," *Acta Astronautica*, vol. 66, no. 1, pp. 220-229, 2010.

[4] J. C. Pemberton and F. Galiber, "A Constraint-Based Approach to Satellite Scheduling," *J. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 57, pp. 101-114, 2001.

[5] J. Frank, A. Jonsson, R. Morris, and D. E. Smith, "Planning and Scheduling for Fleets of Earth Observing Satellites," *Proc. Int'l Symp. Artifical Intelligence, Robotics, Automation and Space (iSAIRAS '05)*, 2001.

[6] P. Soma, S. Venkateswarlu, S. Santhalakshmi, T. Bagchi, and S. Kumar, "Multi-Satellite scheduling using Genetic Algorithms," *Proc. Int'l Symp. Space Mission Operations and Ground Data Systems (SpaceOPS '04)*, 2004.

[7] N. Zufferey, P. Amstutz, and P. Giaccari, "Graph Colouring Approaches for a Satellite Range Scheduling Problem," *J. Scheduling*, vol. 11, no. 4, pp. 263-277, 2008.

[8] J. Wang, J. Li, and Y. Tan, "Study on Heuristic Algorithm for Dynamic Scheduling Problem of Earth Observation Satellites," *Proc. 8th ACIS Int'l Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD '07)*, vol. 1, pp. 9-14, 2007.

[9] S. C. Billups, "Final Rrport of the Ucdhsc Mathematics Clinic Satellite Mission Scheduling with Dynamic Tasking," University of Colorado at Denver and Health Sciences Center, Mathematics Department, 2005.

[10] D. Y. Liao and Y. T. Yang, "Imaging Order Scheduling of an Earth Observation Satellite," *IEEE Trans. Systems, Man and Cybernetics-Part C: Applications and Reviews*, vol. 37, no. 5, pp. 794-802, 2007.

[11] J. Wang, X. Zhu, J. Zhu, M. Ma, and L. T. Yang. "DMTRH: A Real-Time Scheduling Algorithm for Multiple Earth Observation Satellites," *Proc. 14th IEEE Int'l Conf. High Performance Computing and Communications (HPCC '12)*, pp. 673-680, 2012.

[12] J. Wang, X. Zhu, J. Zhu, and M. Ma. "Emergency Scheduling of Multiple Imaging Satellites with Dynamic Merging," *Proc. 12th Int'l Conf. Space Operations (SpaceOPS '12)*, 2012.

[13] T. Tsuchiya, Y. Kakuda and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," *Proc. 2nd Int'l Workshop on Real-Time Computing Systems and Applications (RTCSA '95)*, pp. 197-202, 1995.

[14] S. Ghosh, R. Melhem and D. Mossé, "Fault-Tolerance Through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 3, pp. 272-284, 1997.

[15] R. Al-Omari, A. K. Somani and G. Manimaran, "Efficent Overloading Technique for Primary-Backup Scheduling in Real-Time Systems," *J. Parallel and Distributed Computing*, vol. 64, no. 5, pp. 629-648, 2004.

[16] X. Zhu, C. He, R. Ge, and P. Lu. "Boosting Adaptivity of Fault-Tolerant Scheduling for Real-Time Tasks with Service Requirements on Clusters," *J. Systems and Software*, vol. 84, no. 10, pp. 1708-1716, 2011.

[17] E. Bensana, G. Verfaillie, J. C. Agnese, N. Bataille and D. Blumestein,"Exact and Inexact Methods for the Daily Management of an Earth Observation Satellite," *Proc. Int'l Symp. Space Mission Operations and Ground Data Systems (SpaceOPS '96)*, pp. 507-514, 1996.

[18] M. Vasquez and J. K. Hao, "A 'Logic-Constrained' Knapsack Formulationand a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite," *Journal of Computational Optimization and Applications*, vol. 20, no. 2, pp. 137-157, 2001.

Authorized licensed use limited to: Auburn University. Downloaded on November 17,2026 at 22:57:52 UTC from IEEE Xplore. Restrictions apply.