Fault-Tolerant Scheduling for Real-Time Tasks on Multiple Earth-Observation Satellites

Xiaomin Zhu, *Member, IEEE*, Jianjiang Wang, Xiao Qin, *Senior Member, IEEE*, Ji Wang, Zhong Liu, *Member, IEEE*, and Erik Demeulemeester

Abstract—Fault-tolerance plays an important role in improving the reliability of multiple earth-observing satellites, especially in emergent scenarios such as obtaining photographs on battlefields or earthquake areas. Fault tolerance can be implemented through scheduling approaches. Unfortunately, little attention has been paid to fault-tolerant scheduling on satellites. To address this issue, we propose a novel dynamic fault-tolerant scheduling model for real-time tasks running on multiple observation satellites. In this model, the primary-backup policy is employed to tolerate one satellite's permanent failure at one time instant. In the light of the fault-tolerant model, we develop a novel *f*ault-*t*olerant *s*atellite *s*cheduling algorithm named FTSS. To improve the resource utilization, we apply the overlapping technology that includes primary-backup copy overlapping (i.e., PB overlapping) and backup-backup copy overlapping (i.e., BB overlapping). According to the satellites characterized with time windows for observations, we extensively analyze the overlapping mechanism on satellites. We integrate the overlapping mechanism with FTSS, which employs the task merging strategies including primary-backup copy merging (i.e., PB merging), backup-backup copy merging (i.e., BB merging) and primary-primary copy merging (i.e., PP merging). These merging strategies are used to decrease the number of tasks required to be executed, thereby enhancing system schedulability. To demonstrate the superiority of our FTSS, we conduct extensive experiments using the real-world satellite parameters supplied from the satellite tool kit or STK; we compare FTSS with the three baseline algorithms, namely, NMFTSS, NOFTSS, and NMNOFTSS. The experimental results indicate that FTSS efficiently improves the scheduling quality of others and is suitable for fault-tolerant satellite scheduling.

Index Terms—Earth-observation satellite, fault-tolerance, scheduling, primary-backup copy, overlapping, merging

1 Introduction

Earth observation satellites (EOSs) are platforms equipped with optical instruments that orbit the earth to take photographs of specific areas at user requests [1], [2]. Nowadays, EOSs have been widely used for exploring earth's resources, battlefield reconnaissance, natural disaster surveillance, and the like thanks to their unique advantages such as expansive coverage area, long-term surveillance, and unlimited airspace borders [3], [4], [5]. As the number of satellites grows, multiple satellites are commonly designed to operate in a coordinated way to perform missions. Multiple satellites significantly improve system performance, cost, and survivability compared with the traditional single-satellite deployments [6], [7]. Noticeably, real-time applications have been developed and deployed on multiple EOSs, in which the correctness depends not only on the logical results (e.g., photographs imaged by EOSs), but also on the time instants at which these results are delivered. For example, when an earthquake occurs, there is an urgent need for EOSs to obtain 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 [4]. Another example can be easily found in modern military warfares, in which EOSs are critical equipment to obtain first-hand battlefield information [8]. If the battlefield information cannot be acquired in a timely manner, it will negatively affect decision-making. As such, the US government initiated the Operationally Responsive Spacelift (ORS) plan to design and develop quick-response satellites focusing on tactics and battle missions. Up to now, several small modern satellites (e.g., TacSat-1, TacSat-2, TacSat-3, TacSat-4, and ORS-1) with the characteristics of low cost, short manufacture, and launching period have been successfully launched in recent years to meet emerging and persistent warfighter needs in operationally relevant timelines [9].

Apparently, the aforementioned real-time applications must then incorporate inherent high reliability features. Since the EOSs in the above scenarios are employed in disaster surveillance and military battlefields, where each task must be guaranteed to be completed within its deadline even in the presence of some EOSs' runaway or the failures of instruments installed on EOSs due to enemy interference or other unpredictable reasons. Hence, the management and control system of multiple EOSs on the ground must

- *X. Zhu, J. Wang, J. Wang, and Z. Liu are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan 410073, P.R. China. E-mail: {xmzhu, jianjiangwang, wangji, liuzhong}@nudt.edu.cn.*
- *X. Qin is with the Department of Computer Science and Software Engineering, 3101E Shelby Center, Auburn University, Auburn, AL 36849-5347. E-mail: xqin@auburn.edu.*
- *E. Demeulemeester is with the Department of Decision Sciences and Information Management and Research Center for Operations Management, Naamsestraat, 69, BE-3000 Leuven, KU Leuven, Belgium. E-mail: erik.demeulemeester@kuleuven.be.*

Manuscript received 17 Apr. 2014; revised 8 Oct. 2014; accepted 13 Oct. 2014. Date of publication 16 Oct. 2014; date of current version 7 Oct. 2015. Recommended for acceptance by S. S. Vadhiyar. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2014.2363657

ensure its functional and timing correctness even when faults occur on EOSs. Consequently, providing fault-tolerant mechanisms for such a system coupled with multiple EOSs is mandatory owing to the critical nature of tasks executing on EOSs. Furthermore, in these real-time applications, a great mass of observation information and photos on disaster areas and battlefields are badly required; the arrival times and the number of these real-time earth-observation tasks requested by different users are uncertain, thereby making it a great challenge to develop effective fault-tolerant mechanisms on EOSs.

Scheduling has been regarded as an efficient approach to achieving high performance (e.g., boosting utilization of satellite resources, providing quick timing response, completing more tasks, etc.) in satellite imaging. Task scheduling on EOSs deals with the problem of mapping earth-observation tasks to multiple EOSs to generate schedule decisions satisfying a set of constraints. The most obvious difference between satellite scheduling and other scheduling problems is that satellites are orbiting the earth, which means required areas can only be observed in the satellites' visible ranges (i.e., tasks can only be executed in visible time windows [10]).

Motivation. Due to the significance of scheduling on EOSs, a wide variety of task scheduling algorithms have been proposed in the past decades (see, for example, [1], [2], [11], [12], [13], [14], [15], [16]). Unfortunately, conventional satellite scheduling algorithms, which were mainly developed to optimize task guarantee ratios without addressing the fault tolerance issue, are inadequate for safety-critical real-time applications on EOSs. A handful of fault-tolerant scheduling algorithms have been implemented in parallel and distributed systems such as clusters and grids (see, e.g., [17], [18], [19], [20], [21], [22]). To the best of our knowledge, no work has been done on fault-tolerant scheduling on EOSs. The great challenge is two-fold. First, EOSs' time window constraints and tasks' deadline constraints must be incorporated during fault-tolerant scheduling. Second, many EOSs cannot permanently in contract with a ground control station. Thanks to the development of relay satellites and the space-to-space communication technology among satellites, commands as well as a satellite's status can be quickly transferred in a short time period. Such a technology trend offers ample and promising opportunities to ensure robustness against failures of a satellite running real-time tasks. In this paper, we propose novel fault-tolerant scheduling algorithms for real-time tasks on multiple EOSs. Specifically, our approach focuses on aperiodic, independent, and real-time tasks. We employ the primary-backup model (PB, in short), in which primary-backup overlapping, backup-backup overlapping, the task merging techniques and time window constraints considered.

Contributions. The major contributions of this paper are summarized as follows:

- We establish a satellite fault-tolerant model, which extends the traditional primary-backup fault-tolerant model by incorporating time window constraints of EOSs.
- We analyze task overlapping conditions for the primary-backup overlapping and backup-backup overlapping schemes on multiple EOSs to improve satellite resource utilization.
- We propose three task merging mechanisms while scheduling to decrease the observation number, thus to further enhance satellite resource utilization.
- With the overlapping and merging mechanisms in place, we design a novel dynamic real-time fault-tolerant satellite scheduling algorithm or FTSS to support multiple EOSs.
- We demonstrate that we can design a real-time fault-tolerant scheduling algorithm that is conducive to improve the scheduling performance of the conventional scheduling algorithms on multiple EOSs, where the time window features of satellites are addressed.

The remainder of the paper is organized as follows. The next section reviews related work in the literature. Section 3 formally models the dynamic real-time fault-tolerant scheduling problem on EOSs. This is followed by our proposed FTSS algorithm and the main principles behind it in Section 4. The simulation experiments and performance analysis are given in Section 5. Section 6 concludes the paper with a summary and future work.

2 Related Work

Because of the importance of scheduling in management and control systems of multiple EOSs, a large number of scheduling techniques and algorithms on multiple EOSs have been developed and deployed. Static (off-line) scheduling dominates the scheduling family in the past decades. In a static scheduling algorithm, assignments of tasks to satellites and the time at which the tasks their executions are determined a priori. A static scheduler makes periodic scheduling decisions in one day, one week, or even one month [23].

For instance, Bianchessi et al. [1] investigated an improved Tabu search algorithm to solve the multi-satellite, multi-orbit, and multi-user scheduling problem; they evaluated the quality of the solution with an upper bounding procedure based on the column generation algorithm. After modeling this scheduling problem using a constraint-based language, Frank et al. [13] adopted a stochastic greedy search algorithm on the basis of a heuristic guidance strategy to create solutions. Soma et al. [24] constructed a mathematical model for multi-satellite scheduling; then, they developed an optimization approach by applying a genetic algorithm. Linear programming and heuristics have been investigated to optimize the performance of constellations of small satellites [25]. Zufferey et al. [26] employed the best ingredients of the graph coloring techniques to address the satellite range scheduling problems, proposing the Tabu search and the adaptive memory algorithms. Globus et al. [6], [27] designed an evolutionary algorithm to solve the multi-satellite scheduling problems.

The aforementioned static scheduling schemes lack immediate and dynamic responses, thereby being unable to meet the needs of applications with uncertain arrival times.

With the increase of number of satellites as well as satellite applications, much attention in recent years has been drawn towards dynamic (on-line) scheduling, in which the arrival times of tasks are not known a priori. For example, Wang

et al. [15] investigated an approach to solving the dynamic scheduling problem in the context of distributed imaging satellites. Billups [28] 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 [14], [29] put forward a nominal planning with rolling horizon and lagrangian relaxation technique; they suggested a dynamic policy named FAHA to adjust the nominal planning with updated information. The problem of new image requests that are dynamically added for agile satellites was addressed by Dilkina and Havens [30]. They studied an approach incorporating the advantages of permutation-based search and constraint propagation. However, the above dynamic scheduling schemes were not tailored for real-time applications and; thus, these solutions cannot guarantee the deadlines of real-time tasks. Very recently, we proposed some dynamic scheduling algorithms coupled with the task merging techniques for real-time applications [31], [32], [33]. Our schemes greatly improve the scheduling effectiveness for EOSs.

Little attention has been paid to dynamic fault-tolerant scheduling on EOSs. Fault-tolerant scheduling has been extensively investigated in clusters, grids, and multiple processor systems. Among many fault-tolerant techniques, the Primary-Backup (PB, in short) model is the most popular one. In the PB model, two copies of one task are allocated on two different nodes, where an acceptance test is employed to check the correctness of schedules [34]. For example, two techniques (i.e., deallocation and overloading) were proposed by Ghosh et al. [35] to enhance the schedulability while providing fault tolerance with low overhead. In the overlapping scheme, multiple backup copies can overlap in the same time slot on the same processor. And when a primary copy completes successfully, its corresponding backup copy is reclaimed (i.e., deallocation). Manimaran et al. [36] extended the method described in [35] to tolerate more than one failure by partitioning processors into several groups. In addition, Al-Omari et al. [37] studied a primary-backup overloading technique in which a primary copy can overlap with backup copies of other tasks to further improve the schedulability. Note that these fault-tolerant scheduling approaches belong to the category of passive backup copies (i.e., a backup copy is allowed to execute only if a fault occurs in its primary copy). Besides, the laxity must be at least twice as large as the computation time.

In contrast with the passive backup copy scheme, the active backup copy scheme allows a primary copy to execute simultaneously with its backup copy. Thus, a backup copy can begin to execute even though no fault is detected in its primary copy. Commonly, this scheme is used for the tasks with tight laxity. Tsuchiya et al. [34] proposed a technique in which two copies of each task are concurrently executed with different start times. Yang et al. [38] studied a fault-tolerant scheduling algorithm making the two copies of a task be executed simultaneously to improve schedulability. Al-Omari et al. [19] investigated an adaptive scheme that controls the overlap interval between the primary copy and the backup copy of a task based on the primary-fault probability and the task's laxity. In our previous work, we investigated fault-tolerant scheduling algorithms on clusters [22], [39], [40], where real-time constraints, QoS, and the reliability are generally considered. However, these policies are not suitable for fault-tolerant scheduling on EOSs due to the unique features of EOSs.

In this paper, we concentrate on designing a novel dynamic fault-tolerant scheduling strategy for independent real-time tasks executing on EOSs. Specifically, the PB overlapping, BB overlapping, and task merging techniques are seamlessly integrated to tolerate one satellite's failure at any time instant.

3 System Model

In this section, we introduce the models, notion, and terminology used throughput in this paper. For future reference, we summarize the main notation used in this study in Table 1.

3.1 Task Model

Earth-observation tasks are usually categorized into three types of observation targets, namely, point targets, area targets, and moving targets. A point target is able to be photographed in a single slot by a sensor of a satellite. An area target usually cannot be totally covered by a single observation of a sensor and; therefore, is partitioned into small strips or grids as multiple point targets. When it comes to a moving target, the potential existing area may be predicted in a real-time manner; then, the area can be divided into point targets. As such, we focus on point targets in this study.

We consider a set $T = \{t_1, t_2, \ldots, t_n\}$ of independent non-preemptive real-time tasks arriving aperiodically. Because the Primary-Backup model is employed in our fault-tolerant scheduling scheme, task t_i has two copies (i.e., primary copy t_i^P and backup copy t_i^B that execute on different satellites). Task $t_i \in T$ is modeled as a tuple $t_i = (a_i, d_i, rs_i)$, where a_i, d_i, and rs_i represent task t_i's arrival time, deadline, and resolution requirement, respectively. It is noteworthy that the resolution used in this study refers to spatial resolution.[1]

Let $R = \{r_1, r_2, \ldots, r_m\}$ be a satellite resource set. Resource $r_j \in R$ 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 (OR), respectively [32].

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 $FT^B = (ft_{ij}^B)_{n\times m}$ as the finish time matrix of primary copies and backup copies. $r(t_i^P)$, and $r(t_i^B)$ represents two satellite resources to which t_i^P and t_i^B are allocated, respectively.

We denote $AO_{ij}^P = \{ao_{ij1}^P, ao_{ij2}^P, \ldots, ao_{ijK_{ij}}^P\}$ as a set of available opportunities of t_i^P on resource r_j; AO_i^P is the

1. http://en.wikipedia.org/wiki/Remote_sensing#Data_processing.

TABLE 1
Definitions of Notation

Notation	Definition
t_i	The ith task in the task set $T = \{t_1, t_2, \dots t_n\}$
a_i, d_i, rs_i	t_is arrival time, deadline, and resolution requirement
r_j	The jth satellite resource in the satellite resource set $R = \{r_1, r_2, \dots r_m\}$
du_j, σ_j, s_j	The duration of task execution, field of view angle, slewing pace of r_j
b_j, o_j, as_j	The start-up time, retention time of shutdown, attitude stability time of r_j
msg_j, bor_j	The maximum slewing angle, the best ground observation resolution of r_j
t_i^P, t_i^B	The primary and the backup of task t_i
f_i^P, f_i^B	The finish times of t_i^P and t_i^B
st_{ij}^P, st_{ij}^B	The start time of t_i^P and t_i^B on r_j
ft_{ij}^P, ft_{ij}^B	The finish time of t_i^P and t_i^B on r_j
$r(t_i^P), r(t_i^B)$	The satellite resources to which t_i^P and t_i^B are allocated
ao_{ijk}^P, ao_{ipq}^B	The k available opportunity of t_i^P on r_j and the q available opportunity of t_i^B on r_q
$[ws_{ijk}^P, ws_{ijk}^P], \theta_{ijk}^P$	The time window of observation and the slewing angle of ao_{ijk}^P
$[ws_{ijk}^B, ws_{ijk}^B], \theta_{ijk}^B$	The time window of observation and the slewing angle of ao_{ijk}^B
cao_i^P, cao_i^B	The count of available opportunities of t_i^P and t_i^B
x_{ijk}^P	x_{ijk}^P is "1" if t_i^P is assigned to ao_{ijk}^P; otherwise, x_{ijk}^P is "0"
x_{ipq}^B	x_{ipq}^B is "1" if t_i^B is assigned to ao_{ipq}^B; otherwise, x_{ipq}^B is "0"
z_i^P, z_i^B	The scheduling decision of t_i^P and t_i^B
$or(z_i^P), or(z_i^B)$	The observation resolution of t_i^P and t_i^B employing z_i^P and z_i^B, respectively

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 $ao_{ijk}^P \in AO_i^P$, we have $ao_{ijk}^P = \{[ws_{ijk}^P, we_{ijk}^P], \theta_{ijk}^P\}$, 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. Fig. 1 shows an example of available opportunity ao_{ijk}^P.

Correspondingly, we have $AO_{ip}^B = \{ao_{ip1}^B, ao_{ip2}^B, \dots, ao_{ipQ_{ip}}^B\}$, 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

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

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

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

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

Based on the arrival time and deadline of a task, we can further determine its valid available opportunities. That is:

$$ao_{ijk}^P = \begin{cases} \{[a_i, we_{ijk}^P], \theta_{ijk}^P\}, & \text{if } ws_{ijk}^P < a_i, a_i + du_j \le we_{ijk}^P, \\ \{[ws_{ijk}^P, d_i], \theta_{ijk}^P\}, & \text{if } we_{ijk}^P > d_i, d_i - du_j \ge ws_{ijk}^P. \end{cases} \tag{5}$$

Again, the available opportunities of a backup copy can be adjusted as (5).

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 set to "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, $r(t_i^P) = j \Leftrightarrow \exists k \,|\, x_{ijk}^P = 1$ and $r(t_i^B) = p \Leftrightarrow \exists q \,|\, x_{ipq}^B = 1$. 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,

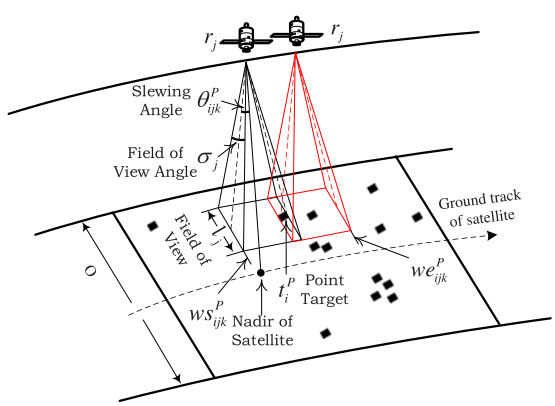


Fig. 1. An example of available opportunity ao_{ijk}^P.

$z_i^B \in Z_i^B$ is a scheduling decision of t_i^B. The observation resolution of t_i^P employing the scheduling decision t_i^P is $or(z_i^P)$ that is computed from (6) below:

$$or\left(z_i^P\right) = \frac{bor_j \cdot L_{ijk}^P}{H_j}, \tag{6}$$

$L_{ijk}^P = (H_j + R)\cos\theta_{ijk}^P - \sqrt{R^2 - (H_j + R)^2 sin^2\theta_{ijk}^P}$, 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) $f_i^P \le d_i$ and $f_i^B \le d_i$; 2) $or(z_i^P) \le rs_i$ and $or(z_i^B) \le rs_i$.

3.2 Fault Model

The fault model used in our study includes the following reasonable assumptions.

- Failures on satellites are transient or permanent, and independent. In other words, a fault occurred on one satellite will not affect another satellite.
- Since the probability that two satellites fail simultaneously is extremely small, we assume that only one satellite fails at any 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 any known failed satellite.

3.3 Scheduling Objectives

The primary objective in this study is to accommodate as many tasks submitted by users as possible while adopting the PB fault-tolerant model. Recall that a primary copy is successfully allocated only if its corresponding backup copy is be scheduled. In other words, if there is no feasible schedule for a backup copy, its primary copy will have to be canceled. The overarching scheduling goal formally delineated in the following expression is to maximize the number of accepted primary copies under timing constraints:

$$\max_{t_i^P \in T, r_j \in R} \left\{ \sum_{j=1}^{m} \sum_{i=1}^{n} \sum_{k=1}^{K_{ij}} x_{ijk}^P \right\}. \tag{7}$$

Another scheduling objectives (See Eq. (8)) is to obtain the maximal observation resolution benefit (i.e., minimize the observation resolutions of all accepted tasks under timing constraints)

$$\min_{t_i^P \in T, t_i^B \in T, r_j \in R} \left\{ \frac{ORP + ORB}{CPB} \right\}, \tag{8}$$

where ORP equals $\sum_{j=1}^{m}\sum_{i=1}^{n}\sum_{k=1}^{K_{ij}} x_{ijk}^P or(z_i^P)$, ORB is equal to $\sum_{p=1}^{m}\sum_{i=1}^{n}\sum_{q=1}^{Q_{ip}} x_{ipq}^B or(z_i^B)$, and CPB is $\sum_{j=1}^{m}\sum_{i=1}^{n}\sum_{k=1}^{K_{ij}} x_{ijk}^P + \sum_{p=1}^{m}\sum_{i=1}^{n}\sum_{q=1}^{Q_{ip}} x_{ipq}^B$.

Eqs. (7) and (8) suggest that our fault-tolerant scheduling strategy has to aim at accommodating a large number of tasks within timing constraints and the resources offering better observation resolution should be chosen in task scheduling.

4 Overlapping Design and Analysis

To efficiently improve the schedulability by boosting the utilization of satellite resources, we 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.

Let us introduce five basic properties to facilitate the presentation of our overlapping mechanism.

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. Thus, we have*

$$\forall t_i \in T, r\left(t_i^P\right) \neq r\left(t_i^B\right). \tag{9}$$

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 encounters a fault and fails to tolerant the fault.

Recall that backup copies may be active or passive. We adopt the passive mode in our scheme, because two different satellites fly with some distance between them and the probability that they take the same photograph at the same time is very slim. Therefore, we have the following property.

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.,*

$$\forall t_i \in T, st_{ip}^B > ft_{ij}^P, j \neq p. \tag{10}$$

Property 3. *If t_i^P finishes successfully within its deadline, then the time slot reserved 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. That is,*

$$\begin{gathered} \forall t_i^P \in T, r_j \in R, ao_{ijk}^P \in AO_{ij}^P, \\ st_{ij}^P \in \left[ws_{ijk}^P, we_{ijk}^P - du_j\right], \theta_{ijk}^P \in [-msg_j, msg_j], \end{gathered} \tag{11}$$

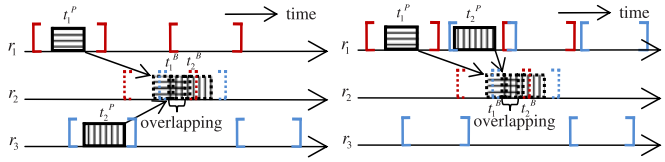
and

$$\begin{gathered} \forall t_i^B \in T, r_p \in R, ao_{ipq}^B \in AO_{ip}^B, \\ st_{ip}^B \in \left[ws_{ipq}^B, we_{ipq}^B - du_j\right], \theta_{ipq}^B \in [-msg_p, msg_p]. \end{gathered} \tag{12}$$

Property 4 indicates that the observation resolution of each task must be better than or equal to the users' resolution requirements. Property 4 leads 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. Thus, we have*

$$\forall t_i^P \in T, t_i^B \in T, or(z_i^P) \le rs_i \text{ and } or(z_i^B) \le rs_i. \tag{13}$$



(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.

4.1 BB Overlapping

To improve the resource utilization, we 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.

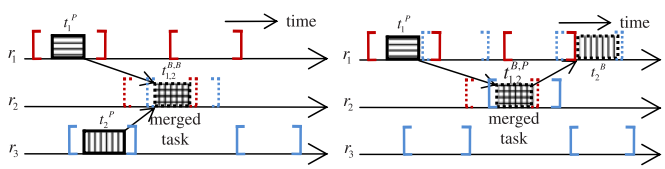
We observe from Fig. 2a that t_1^B and t_2^B are overlapping with each another, and the reason is four-fold.

- If resources r_1, r_2, and r_3 are fault-free, t_1^P and t_2^P will be finished successfully. Thus, there is no need to execute t_1^B and t_2^B any more. Therefore, t_1^B and t_2^B can overlap.
- If resource r_1 has a fault, t_1^P will fail. In this case, t_1^B has to be executed. According to the assumption in Section 3.2, r_2 and r_3 will not fail and; 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 reserved for t_2^B will be reclaimed, avoiding any conflicts 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 in resource r_3, t_2^P cannot be successfully finished. There is no timing conflicts between t_1^B and t_2^B.
- 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 are performing, then t_1^P and t_2^P can be finished successfully. Thus, the overlapping between 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. Fig. 2b shows an unfeasible allocation example.

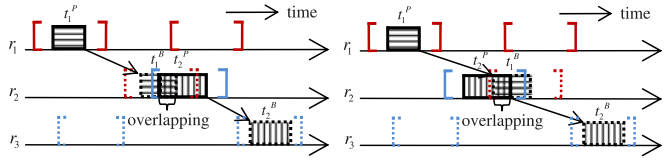
Fig. 2b illustrates that if resource r_1 encounters a fault, t_1^B and t_2^B must be executed. Due to the overlapping of t_1^B and t_2^B, there exists a timing conflict between t_1^B and t_2^B. Consequently, we obtain the following property.

Property 6. *If primary copies $t_i^P, t_j^P, \ldots, t_k^P$ have been allocated to the same resource, then their corresponding backup copies $t_i^B, t_j^B, \ldots, t_k^B$ cannot have overlapping part in timing.*



(a) BB Merging (b) PB Merging

Fig. 3. Some examples of merging.



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

Fig. 4. Examples of PB overlapping.

According to the features of satellite observation (i.e., a scene imaged by a satellite sensor is with a certain size), adjacent targets in a scene can be observed at the same time [41]. This unique characteristic motivates us to merge tasks while applying the overlapping strategy to improve the schedulability.

Now, we give an example (see Fig. 3a) to show how to merge tasks.

In Fig. 3a, $t_{1,2}^{B,B}$ represents a merged task formed from t_1^B and t_2^B. If a fault occurs 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). The merging constraints is discussed in Section 4.4.

Additionally, if multiple primary copies are assigned to the same resource, their corresponding backup copies cannot be merged. The PB merging technique is described in Section 4.2.

4.2 PB Overlapping

A second overlapping mechanism 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 feasible PB overlapping is shown in Fig. 4a.

Fig. 4a shows that t_1^B can overlap with t_2^P. Let us consider the following four cases.

- If resources r_1, r_2 and r_3 encounter no failure, 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 a fault occurs in resource r_1, 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. Thus, t_1^B and t_2^P are able to overlap with 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 time conflict between t_1^B and t_2^P. Hence, the overlapping of t_1^B and t_2^P is acceptable.

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. 4b.

Suppose that resource r_1 encounters a fault, t_1^B must execute. 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.

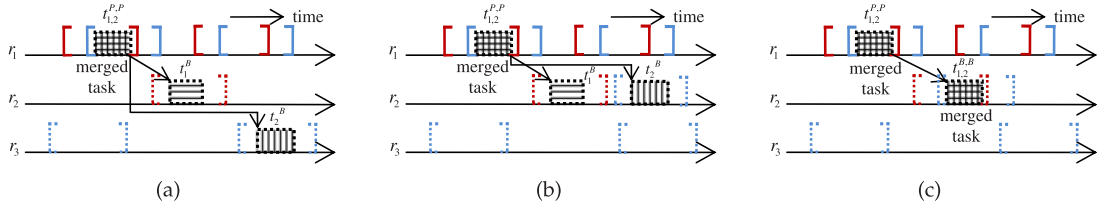


Fig. 5. Examples of successful PP merging.

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.,*

$$st_{ip}^P > st_{jp}^B. \tag{14}$$

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 $t_j^B, \ldots, t_k^B$ overlap, t_i^B and $t_j^P, \ldots, t_k^P$ cannot be allocated to the same resource.*

However, based on the example in Fig. 4a, 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. 3b.

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 regardless of any failure on r_1 or r_2. Merged task $t_{1,2}^{B,P}$ is formed from t_1^B and t_2^P; we analyze the merging constraints in Section 4.4.

4.3 PP Merging

From the aforementioned analysis of the overlapping technologies, we show that primary copies cannot be overlapped because overlapping primary copies fails to tolerate failures. Nevertheless, multiple primary copies can be merged while conducting fault-tolerant scheduling besides the PB merging and BB merging discussed in the PB and BB overlapping. Fig. 5 depicts three PP merging examples.

Fig. 5a illustrates that if t_1^P merges with t_2^P, their corresponding backup copies t_1^B and t_2^B might be allocated to two other different resources. Thus, we can easily conclude that no matter which resource encounters a fault, t_1 and t_2 can be successfully finished.

If resource r_1 fails (see Fig. 5b), t_1^B and t_2^B can run successfully. If resource r_2 encounters a fault, $t_{1,2}^{P,P}$ also can successfully finish. Thus, we can conclude that if t_1^P merges with t_2^P, t_1^B and t_2^B can be allocated to the same resource except for $r(t_{1,2}^{P,P})$ and at the same time t_1^B and t_2^B have no overlapping.

Fig. 5c clearly shows that if t_1^P merges with t_2^P, it is also feasible to merge t_1^B with t_2^B, and $t_{1,2}^{P,P}$ and $t_{1,2}^{B,B}$ need to be allocated to different resources.

On the other hand, if t_1^B and t_2^B has to employ the overlapping method rather than merging, it cannot realize fault-tolerance. Fig. 6 depicts an unfeasible example of PP merging.

We observe from Fig. 6 that if resource r_1 encounters a fault, t_1^B and t_2^B will start to execute. t_1^B and t_2^B have a time conflict during their execution, thereby violating the fault-tolerance requirements.

Theorem 1. *Multiple primary copies $t_i^P, \ldots, t_j^P, \ldots, t_k^P$ can merge while making fault-tolerant scheduling and their backup copies $t_i^B, \ldots, t_j^B, \ldots, t_k^B$ cannot overlap when allocating these backup copies.*

Proof. Since $t_{i,\ldots,j,\ldots,k}^{P,\ldots,P,\ldots,P}$ is a merged task, there exists no timing conflict of $t_i^P, \ldots, t_j^P, \ldots, t_k^P$ when executing. Besides, suppose $t_i^B, \ldots, t_j^B, \ldots, t_k^B$ can overlap, if $r(t_{i,\ldots,j,\ldots,k}^{P,\ldots,P,\ldots,P})$ fails, $r(t_{i,\ldots,j,\ldots,k}^{B,\ldots,B,\ldots,B})$ has to execute, but it is impossible to finish all of them successfully because there exists overlapping in them. It means that the assumption is incorrect, which completes the proof for this Theorem. □

4.4 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 (e.g., X and Y may be P or B). The following four constraints must be met.

First, t_a^X and t_b^Y must have an overlapped time window. Thus, we have

$$\begin{aligned} &\forall t_a^X \in T, t_b^Y \in T, r_j \in R, ao_{ajk}^X \in AO_{aj}^X, ao_{bjq}^Y \in AO_{bj}^Y, \\ &we_{ajk}^X > ws_{bjq}^Y, \text{ if } we_{bjq}^Y > we_{ajk}^X, \\ &we_{bjq}^Y > ws_{ajk}^X, \text{ if } we_{ajk}^X > we_{bjq}^Y. \end{aligned} \tag{15}$$

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

$$\min\{we_{ajk}^X, we_{bjq}^Y\} - \max\{ws_{ajk}^X, ws_{bjq}^Y\} \geq du_j\,. \tag{16}$$

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,

$$\left|\theta_{ajk}^X - \theta_{bjq}^Y\right| \leq \sigma_j\,. \tag{17}$$

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. Thus, we have the following constraint:

$$or\left(z_{ab}^{XY}\right) \leq rs_a, \text{ and } or\left(z_{ab}^{XY}\right) \leq rs_b\,. \tag{18}$$

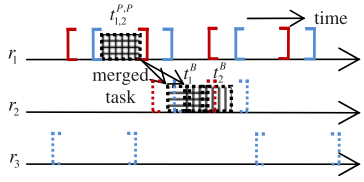


Fig. 6. An example of unfeasible PP merging.

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

$$tw_{ab,j,kq}^{XY} = |\max\{ws_{ajk}^X, ws_{bjq}^Y\}, \min\{we_{ajk}^X, we_{bjq}^Y\}|. \quad (19)$$

Also, the new slewing angle is set to be:

$$\theta_{ab,j,kq}^{XY} = (\theta_{a,j,k}^X + \theta_{b,j,q}^X)/2. \quad (20)$$

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

$$ao_{ab,j,kq}^{XY} = \{tw_{ab,j,kq}^{XY}, \theta_{ab,j,kq}^{XY}\}. \quad (21)$$

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.

The aforementioned merging of tasks only considers two tasks. However, it can be easily extended to merge multiple tasks only by repeating the merging operations of two tasks under the constraints (15)-(18).

The main differences between the overlapping and merging techniques are summarized as follows:

- If multiple tasks merge into one task, then these tasks are treated as a single task. In contrast, when multiple tasks are overlapping, these tasks cannot be envisioned as a single task; moreover, only one of the overlapped tasks can be executed according to our fault model.
- The overlapping technique is applied to decrease occupied time slots, whereas the merging technique aims to reduce the number of tasks.
- Multiple primary copies can be merged. On the other hand, multiple primary copies are unable to overlap one another.
- The constraints of overlapping are different from those of merging.

5 Fault-Tolerant Satellite Scheduling Algorithm FTSS

In this section, we present the novel *f*ault-*t*olerant *s*atellite *s*cheduling algorithm for real-time, independent, aperiodic tasks running on multiple EOSs. FTSS judiciously considers observation resolution, resource utilization, schedulability, and fault-tolerance.

5.1 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.,*

$$p_{i-1,i,j}^{XP} = o_j + \frac{\left|\theta_{i-1,j,k}^X - \theta_{i,j,q}^P\right|}{s_j} + as_j + b_j. \quad (22)$$

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

$$r_{ij}^P = \max\{p_{i-1,i,j}^{XP} + ft_{i-1,j}, a_i\}. \quad (23)$$

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.,

$$f_i^P < d_i. \quad (24)$$

In the first way, constraints (15)-(18) must be satisfied to guarantee that t_i^P is able to merge with a task (or a merged task). est_{ijk}^P is equal to the start time of task (or merged task) t_a^X to be merged with. That is,

$$est_{ijk}^P = st_{aj}^X. \quad (25)$$

For the second way, t_i^P can overlap with a backup copy or overlap with a merged task that cannot derive from any primary copies, i.e., if t_i^P can overlap with a merged task $t_{i_1,i_2,\ldots,i_k}^{X_1,X_2,\ldots,X_k}$,

$$X_1 \neq P \text{ and } X_2 \neq P \text{ and } \cdots \text{ and } X_k \neq P. \quad (26)$$

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:

$$est_{ijk}^P = \max\{r_{ij}^P, st_{aj}^X + \varepsilon\}, \quad (27)$$

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 must be calculated. Without loss of generality, before computing est_{ijk}^P, we assume that $t_{i_1}, t_{i_2}, \ldots, t_{i_q}$ have been allocated to the kth available opportunity on resource r_j. To simplify the symbol, we refer these copies to be either primary copies or backup copies; these copies can be either single tasks or merged tasks. The idle time slots on ao_{ijk}^P are $[ws_{ijk}^P, st_{i_1}], [ft_{i_1}, st_{i_2}], \ldots, [ft_{i_{q-1}}, st_{i_q}], [ft_{i_q}, we_{ijk}^P]$. 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:

$$\max\{r_{ij}^P, ft_{i_k} + p_{i_k,i,j}^{XP}\} + du_j \leq st_{i_{k+1}}. \quad (28)$$

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

$$est_{ijk}^P = \max\{r_{ij}^P, ft_{i_k} + p_{i_k,i,j}^{XP}\}. \quad (29)$$

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

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

Algorithm 1. Primary Copies Scheduling in FTSS

foreach *new task* t_i, *schedule its primary copy* t_i^P **do**
  $allocationTag \leftarrow$ FALSE; $mergeTag \leftarrow$ FALSE;
  foreach *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 $allocationTag \neq TRUE$ **then**
      POverlapping()
      PInserting()
    if $allocationTag == TRUE$ and $mergeTag == FALSE$ **then**
      $est_{ij}^P \leftarrow \min\{est_{ij}^{P'}, est_{ij}^{P''}\}$;
    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;
  Sort the allocation schemes in $aScheme_i^P$ by t_i^P's start time in ascending order;
  $aScheme_i^P(u) \leftarrow$ the former $u\%$ schemes in $aScheme_i^P$;

Algorithm 2. Function PMerging()

foreach *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;
    $est_{ijk}^P \leftarrow st_{aj}^X$;
    Calculate t_i^P's finish time f_i^P;
    if $f_i^P < d_i$ **then**
      $mergeTag \leftarrow$ TRUE; $allocationTag \leftarrow$ TRUE;
      break;

Algorithm 3. Function POverlapping()

foreach *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;

Now, we evaluate the time complexity of primary allocation in FTSS as shown below. Suppose we consider scheduling a primary copy of task i. Let m denote the number of resources. Let K_{ij} denote the number of available opportunities of t_i^P on resource r_j. Let $N_{tw}(ijk)$ denote the number of existing allocated tasks in tw_{ijk}^P. Let K_i denote the maximum number of K_{ij} on all resources. Let N_{tw} denote the maximum number of $N_{tw}(ijk)$ within all the available opportunities on all resources.

Algorithm 4. Function PInserting()

foreach 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;

Theorem 2. *The time complexity of primary allocation in FTSS is* $O(mK_iN_{tw})$.

Proof. In the function PMerging(), the time complexity of calculating the earliest start time of t_i^P in one available opportunity ao_{ijk}^P on r_j is $O(N_{tw}(ijk))$. Then, for all the available opportunities on r_j, the time complexity is $O(K_{ij}\max\{N_{tw}(ijk)\})$, i.e. the time complexity of PMerging(). The time complexities of function POverlapping() and PInserting() both are $O(K_{ij}\max\{N_{tw}(ijk)\})$. The calculation process is similar to that of PMerging(). For all the resources, the time complexity in the worst situation is $O(K_iN_{tw})$. Therefore, the overall time complexity is calculated as follows: $O(mO(K_iN_{tw} + K_iN_{tw} + K_iN_{tw})) = O(mK_iN_{tw})$. □

5.2 Backup Copies Scheduling

We schedule a backup copy, whose primary copy has been allocated, as late as possible to save earlier time slots for other later-arriving tasks to meet their deadlines. Additionally, idle time slots that are 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 (15)-(18) must be satisfied to guarantee t_i^B being able to be merged 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.,

$$lst_{ipq}^B = st_{bp}^X. \tag{30}$$

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

$$lst_{ipq}^B = \begin{cases} \min\{we_{ipq}^B, d_i\} - du_j, & \text{if } lts = \text{idle} \vee ol(t_i^B), \\ st_{bp}^X - p_{ibp}^{BX} - du_j, & \text{if } ts = \text{idle} \vee ol(t_i^B), \end{cases} \tag{31}$$

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.

Algorithm 5. Backup Copies Scheduling in FTSS

1 **foreach** *new task t_i whose primary copy t_i^P has not been rejected, schedule its backup copy t_i^B* **do**
2 $allocationTag \leftarrow$ FALSE; $value \leftarrow +\infty$;
3 **foreach** *allocation scheme of t_i^P in $aScheme_i^P(u)$* **do**
4 $allocationTag' \leftarrow$ FALSE; $mlst \leftarrow 0$; $nmlst \leftarrow 0$;
5 **foreach** *resource r_j except $r(t_i^P)$ in the system* **do**
6 Calculate t_i^B's available opportunities AO_{ip}^B;
7 Calculate t_i^B's valid available opportunities as (5);
8 **BMerging**();
9 **if** $allocationTag \neq TRUE$ and $mlst == 0$ **then**
10 **BOverlappingOrInserting**();
11 **if** $allocationTag' == TRUE$ *and* $mlst \neq 0$ **then**
12 Put allocation scheme with maximal $mlst$ of t_i^B into set $aScheme_i^B$;
13 **else if** $allocationTag' == TRUE$ *and* $mlst == 0$ **then**
14 Put allocation scheme with maximal $nmlst$ of t_i^B into set $aScheme_i^B$;
15 **if** $allocationTag == FALSE$ **then**
16 Reject t_i^P and t_i^B;
17 **foreach** *primary copy allocation scheme in $aScheme_i^P(u)$ and the corresponding backup copy allocation scheme in $aScheme_i^B$* **do**
18 Calculate $value_i = \frac{st_i^P}{st_i^B}/(or(z_i^P) + or(z_i^B))$;
19 **if** $value_i < value$ **then**
20 Select the current task allocation scheme;

Algorithm 6. Function BMerging()

1 **foreach** *valid available opportunity ao_{ipq}^B* **do**
2 **if** *t_i^B can merge with a task (or a merged task) t_b^X allocated in tw_{ipq}^B* **then**
3 Merge t_i^B with t_b^X;
4 $lst_{ipq}^B \leftarrow st_{bp}^X$;
5 **if** $lst_{ipq}^B > nmlst$ **then**
6 $mlst \leftarrow lst_{ipq}^B$;
7 $allocationTag \leftarrow$ TRUE; $allocationTag' \leftarrow$ TRUE;
8 **break;**

Then, we evaluate the backup allocation's time complexity in FTSS. We consider scheduling a backup copy of task i. Let $N_a(i)$ denote the number of allocation schemes of t_i^P in $aScheme_i^P(u)$. Let Q_{ip} denote the number of available opportunities of t_i^B on resource r_p. Let $N_{tw}(ipq)$ denote the number of existing allocated tasks in tw_{ipq}^B. Let Q_i denote the maximum number of Q_{ip} on all resources. Let N'_{tw} denote the maximum number of $N_{tw}(ipq)$ within all the available opportunities on all resources.

Theorem 3. *The time complexity of primary allocation in FTSS is* $O(mN_a(i)Q_iN'_{tw})$.

Proof. As the similar calculation process in Theorem 2, the time complexities of BMerging() and BOverlappingOrInserting() both are $O(Q_{ip}\max\{N_{tw}(ipq)\})$. The time complexity for examining all the resources is $O(mQ_iN'_{tw})$. For all the allocation schemes of t_i^P in $aScheme_i^P(u)$, the time complexity is $O(mN_a(i)Q_iN'_{tw})$. For lines 17-20 in Algorithm 5, the time complexity is $O(N_a(i))$. As a result, the time complexity of the backup allocation in FTSS is calculated as follows: $O(mN_a(i)Q_iN'_{tw} + N_a(i)) = O(\max\{mN_a(i)Q_iN'_{tw}, N_a(i)\}) = O(mN_a(i)Q_iN'_{tw})$. □

6 Performance Evaluation

We evaluate in this section the performance of the proposed FTSS algorithm. To demonstrate the performance improvements gained by FTSS, we quantitatively compare it with three baseline algorithms, namely, a *f*ault-*t*olerant *s*atellite *s*cheduling algorithm without *m*erging (NMFTSS); a *f*ault-*t*olerant *s*atellite *s*cheduling algorithm without *o*verlapping (NOFTSS); a *f*ault-*t*olerant *s*atellite *s*cheduling algorithm without *m*erging and *o*verlapping (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:

- Guarantee Ratio (GR) defined as: $GR =$ Total number of tasks guaranteed to meet their deadlines/ Total number of tasks $\times$ 100%;
- Observation Resolution used to test the average observation resolution of accepted tasks.

6.1 Simulation Method and Parameters

To validate the performance improvements of FTSS, the targets are randomly generated in the area: latitude $-$30-60 degree and longitude 0-degree. The number of targets are 200, 400, 600, 800, 1,000, 1,200, respectively. According to extensive literature ([12], [23], [42], [43], [44]), ten sensors on different satellites are considered in this paper. The parameters of the sensors are presented in Table 2, and the orbit models of satellites are obtained from the satellite tool kit or STK,[2] where the parameters with (*) denote the designed values based on the previous literature.

- 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 $[a, b], a_0 = 0$;

2. http://www.agi.com/.

TABLE 2
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

- The deadline d_i is expressed as $d_i = a_i + Deadline$, where $Deadline \sim N(baseDeadline, baseDeadline/10)$.

Table 3 summarizes the simulation parameters and their values. The STK presents the running status of earth-observation system composed by 10 satellites listed in Table 2. Besides, the random synthetic tasks generated by our simulation parameters are shown on the earth (Target i denotes the ith task). Fig. 7 shows the 3D graphic when 100 tasks are generated.

Algorithm 7. Function BOverlappingOrInserting()

```
1 foreach valid available opportunity ao^B_ipq do
2   if t^B_i can be accommodated by overlapping or inserting
    then
3     Calculate t^B_i's latest start time lst^B_ipq by (33);
4     Calculate t^B_i's observation resolution or(z^B_i) as (6);
5     if or(z^B_i) ≤ rs_i then
6       allocationTag ← TRUE; allocationTag' ← TRUE;
7       if lst^B_ipq > nmlst then
8         nmlst ← lst^B_ipq;
9       break;
```

In our experiments, we employ the "Once Tuning One Parameter (OTOP)" method, which has been widely applied in simulation experiments reported in the literature (see, for example, [29], [31], [40]). In each experiment, we change a single parameter while keeping all the other parameters fixed. Tuning one parameter at a time allows us to clearly observe impacts of the parameter on performance of our scheduling system.

6.2 Performance Impact of the Number of Tasks

In this experiment, we investigate the performance impact of the number of tasks that increases from 200 to 1,200 with an increment of 200. Fig. 8 illustrates the performances of FTSS, NMFTSS, NOFTSS and NMNOFTSS in terms of guarantee ratio and observation resolution.

Fig. 8a shows that with the increase of the number of tasks, the guarantee ratios of all the algorithms decrease. This trend is reasonable, because of given limited resources, increasing the number of tasks makes the tested system heavily loaded, which in turn reduces the guarantee ratios. Besides, Fig. 8a shows that the guarantee ratio of NMNOFTSS is the lowest because neither the overlapping technique nor the task merging strategies are employed in the scheduling process of 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, which boosts the resource utilization. The number of tasks decreases by the introduction of the task merging strategy. Consequently, the guarantee ratio of FTSS is the highest among all the evaluated solutions. We can observe from Fig. 8a that NMFTSS and NOFTSS outperform NMNOFTSS.

TABLE 3
Parameters for Simulation Studies

Parameter	Value(Fixed)-(Varied)
Number of Tasks	(1,000)-(200, 400, 600, 800, 1,000, 1,200)
Number of Resources	(10)
$intervalTime$(h)	([20, 40])-([0, 20], [20, 40], [40, 60], [60, 80], [80, 100])
u	(50)-(10, 20, 30, 40, 50, 60, 70, 80)
$baseDeadline$(h)	(30)-(20, 30, 40, 50, 60, 70)

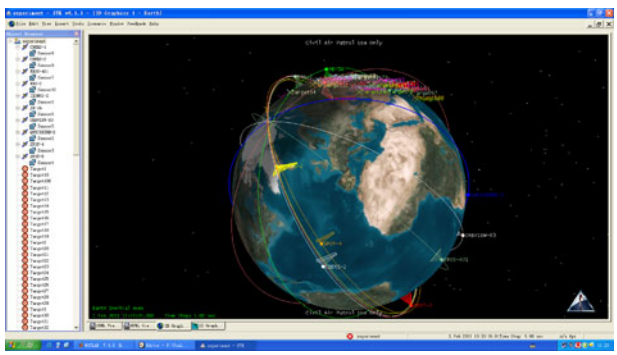


Fig. 7. STK 3D graphics.

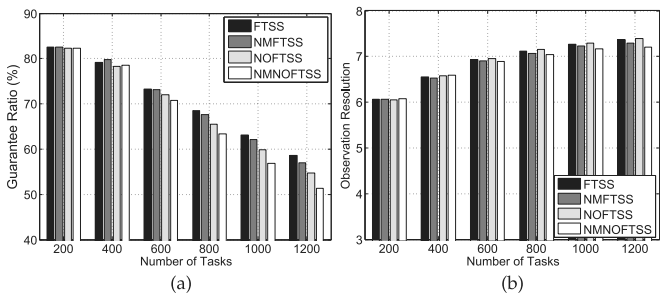


Fig. 8. Performance impact of the number of tasks.

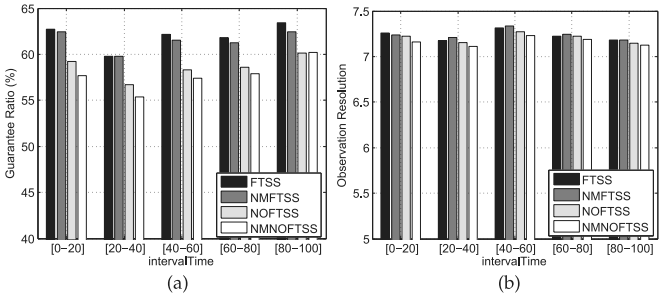


Fig. 9. Performance impact of arrival rate.

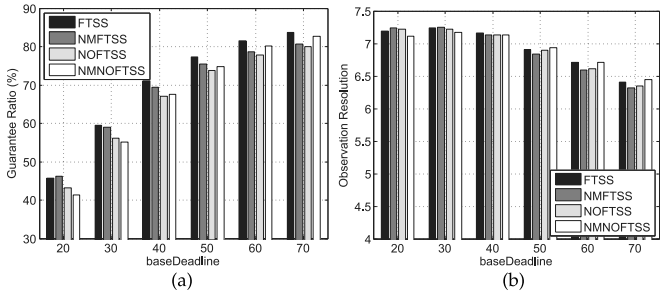


Fig. 10. Performance impact of deadline.

Besides, the guarantee ratio of NMFTSS is higher than that of NOFTSS, because the constraints of task merging are stringent, which makes low guarantee ratio of NOFTSS.

Fig. 8b demonstrates that the observation resolutions of the four algorithms go up with the increasing number of tasks. When the resources are limited, the increase of the number of tasks results in heavy load, which makes the system downgrade observation resolutions to accept more tasks. Note that all the observation resolutions of accepted tasks satisfy user requirements. Hence, the observation resolutions ascend. We observe from Fig. 8b that when the system workload is light (e.g., the number of tasks is less than 400), the observation resolution of FTSS is better than that of NMNOFTSS. This result confirms that FTSS can optimize the observation resolution as well as accept more tasks. When the system workload is heavy (the number of tasks 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. Besides, the observation resolution of NMFTSS outperforms NOFTSS. During the of task-merging process, a sensor must adjust its observation angle to cover several targets in one imaging procedure, thereby leading to worse observation resolutions. NMFTSS does not need to adjust satellite sensors' observation angles for covering several targets. Thus, the observation resolution of NMFTSS is better.

6.3 Performance Impact of Arrival Rate

We carry out a group of experiments to observe the impact of arrival rate on the four 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. 9.

Fig. 9a demonstrates that when we increase the *intervalTime* from [0, 20] to [20, 40], the guarantee ratios of all the algorithms noticeably decrease because of two reasons. 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 rapidly arrive, more time windows of tasks can overlap with others, thereby providing more opportunities for task overlapping and merging. On the other hand, when *intervalTime* becomes a little larger (i.e., the system workload is 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.

Fig. 9b shows that the differences among the four algorithms are marginal. The observation resolutions are improved with the increase of *intervalTime*, because a system under light workload can offer good observation resolutions while accommodating more tasks. Additionally, among the four algorithms, NMNOFTSS has the best observation resolutions while the worst guarantee ratio (see Fig. 9a). 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 exhibits good performance in terms of observation resolution.

6.4 Performance Impact of Task Deadline

Now we investigate the impact of the tasks' deadlines on the performance of the four algorithms. The parameter *baseDeadline* varies from 20 to 70 with an increment of 10. Fig. 10 illustrates the experimental results.

Fig. 10a demonstrates that the guarantee ratios of the four algorithms show ascending trends with the increase of *baseDeadline*. The reason is that the deadlines 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

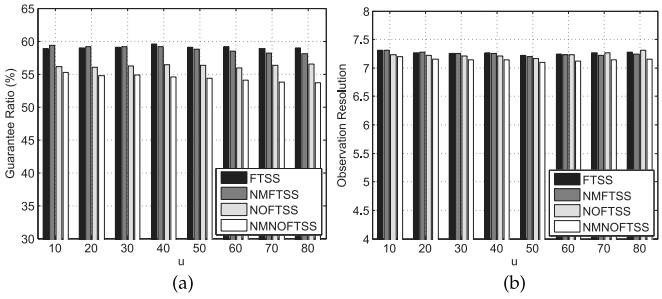


Fig. 11. Performance impact of parameter *u*.

FTSS, the opportunities for task overlapping also increase in the condition that the 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. 10b depicts that the observation resolutions of all the algorithms slightly increase first and; then, the observation resolution drops with the increasing *baseDeadline*. When the deadline becomes a bit looser, the system is still overloaded. In this case, some tasks' observation resolutions must be degraded so that an increasing number of tasks may have feasible schedules. When we further loosen the deadline, the system is able to accept more tasks, thereby offering higher observation resolutions. The observation resolution of FTSS is marginally higher than that of NMNOFTSS (i.e., the average difference is only 0.014623), but FTSS outperforms NMNOFTSS in terms of guarantee ratio.

6.5 Performance Impact of Parameter *u*

We evaluate in this experiment the performance impact of the parameter *u* that guides FTSS to schedule primary copies by yielding multiple candidate allocations for primary copies. The experimental results are shown in Fig. 11.

It is observed from Fig. 11a that the guarantee ratios of all the algorithms are insensitive to parameter *u* increases. We attribute this trend to the limited number (e.g., 10) of resources, which causes only one or two allocation schemes of primary copies to be available. Thus, the available overlapping schemes of backup copies are few; *u* has little impact on the performance of the algorithms. In contrast, when the number of resources is great or the number of tasks is relatively small, the value of *u* has significant impacts. Besides, regardless of the changing value of *u*, FTSS consistently exhibits the best performance.

Fig. 11b depicts that the observation resolutions of all the algorithms almost remain unchanged. This result confirms that parameter *u* has no noticeable impact on system performance, because there are few available overlapping schemes of backup copies due to scarce resources. In addition, the observation resolution of FTSS is consistently higher than that of NMNOFTSS, because FTSS makes an efforts to improve guarantee ratio at the cost of observation resolutions under the condition that the observation resolutions satisfy user requirements.

7 Conclusions and Future Work

In this study, we investigated the problem of fault-tolerant scheduling for real-time tasks on multiple earth observation satellites. The fault-tolerant capability is achieved by applying the primary-backup policy. The scheduling objective is to improve real-time tasks' schedulability and optimize tasks' observation resolutions without violating any fault-tolerant requirements. We first built a scheduling model incorporating the primary-backup policy. Then, we analyzed the task overlapping conditions on multiple EOSs and the constraints of task merging. Next, we proposed a novel fault-tolerant satellite scheduling algorithm called FTSS, which seamlessly integrates task overlapping, task merging, and task inserting. FTSS can efficiently enhance resource utilization and; thus, FTSS improves system performance in terms of guarantee ratios and observation resolutions.

Our FTSS is the first of its kind reported in the literature, because it comprehensively addresses the issue of fault-tolerance, real-time, schedulability, and adaptivity. To evaluate the performance of FTSS, we conducted extensive experiments to compare FTSS with the three baseline algorithms. The experimental results indicate that FTSS significantly improves the scheduling quality of the other solutions. Our experiments confirm that FTSS is conducive to fault-tolerant satellite scheduling in dynamic environments.

There are a few open issues to be addressed in our future studies. First, we will extend our fault-tolerant scheduling model to tolerate multiple satellites' failures. We plan to extend FTSS to develop new scheduling algorithms tolerating multiple failures and for periodic tasks. Second, we will take communication times and task dispatching times into account to enhance scheduling accuracy. Third, we plan to implement and test FTSS in one of our real-world satellite systems.

Acknowledgments

This research was supported by the Program for Changjiang Scholars and Innovative Research Team in University of China Under grant No. IRT13014, the National Natural Science Foundation of China under grant No. 61104180 and No. 71271213.

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," *Eur. J. Oper. Res.*, vol. 177, no. 2, pp. 750–762, 2007.

[2] J. Cordeau and G. Laporte, "Maximizing the value of an earth observation satellite orbit," *J. Oper. Res. Soc.*, vol. 56, no. 8, pp. 962–968, 2004.

[3] W. C. Lin and S. C. Chang, "Hybrid algorithms for satellite imaging scheduling," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, 2005, vol. 3, pp. 2518–2523.

[4] K. J. Zhu, J. F. Li, and 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.

[5] C. D. Jilla and D. W. Miller, "Assessing the performance of a heuristic simulated annealing algorithm for the design of distributed satellite systems," *Acta Astronautica*, vol. 48, no. 5, pp. 529–543, 2001.

[6] A. Globus, J. Crawford, J. Lohn, and A. Pryor, "A comparison of techniques for scheduling fleets of earth-observing," *J. Oper. Res. Soc.*, vol. 56, no. 8, pp. 962–968, 2003.

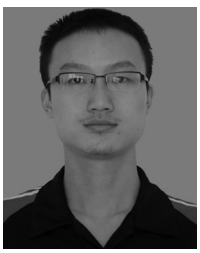
[7] G. B. Shaw, D. W. Miller, and D. E. Hastings, "Generalized characteristics of communication, sensing, and navigation satellite systems," *J. Spacecraft Rockets*, vol. 37, no. 6, pp. 801–811, 2000.

[8] C. Rivett and C. Pontecorvo, "Improving satellite surveillance through optimal assignment of assets," Department of Defense, Australian Government, Defence Sci. Technol. Organisation Tech. Rep.-1488, 2003.
[9] ORS-1 First ORS Satellite Designed To Support Combatant Command Operations. (2012) [Online]. Avaialble: http://www.goodrich.com
[10] J. C. Pemberton and F. Galiber, "A constraint-based approach to satellite scheduling," *J. DIMACS Series Discrete Math. Theoretical Comput. Sci.*, vol. 57, pp. 101–114, 2001.
[11] J. Pemberton, "Towards scheduling over-constrained remote sensing satellites," in *Proc. 2nd Int. NASA Workshop Planning Scheduling Space*, 2000, pp. 84–89.
[12] 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," in *Proc. Int. Symp. Space Mission Operations Ground Data Syst.*, 1996, pp. 507–514.
[13] J. Frank, A. Jonsson, R. Morris, and D. E. Smith, "Planning and scheduling for fleets of earth observing satellites," in *Proc. Int. Symp. Artif. Intell., Robot., Autom. Space*, 2001.
[14] D. Y. Liao and Y. T. Yang, "Satellite imaging order scheduling with stochastic weather condition forecast," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, 2005, vol. 3, pp. 2524–2529.
[15] J. Wang, J. Li, and Y. Tan, "Study on heuristic algorithm for dynamic scheduling problem of earth observation satellites," in *Proc. 8th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, 2007, vol. 1, pp. 9–14.
[16] M. Werner, D. Müller, M. Däumler, J. Richling, and G. Mühl, "Operating system support for distributed applications in real space-time," in *Proc. 5th Int. Conf. Soft Comput. Transdisciplinary Sci. Technol.*, 2008, pp. 469–478.
[17] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *J. Parallel Distrib. Comput.*, vol. 65, no. 8, pp. 885–900, 2005.
[18] W. Luo, J. Li, F. Yang, G. Tu, L. Pang, and L. Shu, "DYFARS: Boosting reliability in fault-tolerant heterogeneous distributed systems through dynamic scheduling," in *Proc. 8th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw., Parallel/Distrib. Comput.*, 2007, pp. 640–645.
[19] R. Al-Omari, A. K. Somani, and G. Manimaran, "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 5, pp. 595-608, 2005.
[20] Q. Zheng, B. Veeravalli, and C. K. Tham, "On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 380–393, Mar. 2009.
[21] Y. S. Hong and H. W. Goo, "A fault-tolerant scheduling scheme for hybrid tasks in distributed real-time systems," in *Proc. 3rd IEEE Workshop Softw. Technol. Future Embedded Ubiquitous Syst.*, 2005, pp. 3–6.
[22] X. Zhu, X. Qin, and M. Qiu, "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 6, pp. 800–812, Apr. 2011.
[23] M. Vasquez and J. K. Hao, "A 'logic-constrained' knapsack formulationand a tabu algorithm for the daily photograph scheduling of an earth observation satellite," *J. Comput. Optim. Appl.*, vol. 20, no. 2, pp. 137-157, 2001.
[24] P. Soma, S. Venkateswarlu, S. Santhalakshmi, T. Bagchi, and S. Kumar, "Multi-satellite scheduling using genetic algorithms," in *Proc. Int. Symp. Space Mission Oper. Ground Data Syst.*, 2004, pp. 1–10.
[25] C. Rivett and C. Pontecorvo, "Improving satellite surveillance through optimal assignment of assets," Australian Government Dep. Defense, Rep. No. DSTO-TR-1488, 2004,
[26] 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.
[27] A. Globus, J. Crawford, J. Lohn, and A. Pryor, "Scheduling earth observing satellites with evolutionary algorithms," in *Proc. Space Mission Challenges Inform. Technol. Conf.*, 2003.
[28] S. C. Billups, *Final Report of the UCDHSC Mathematics Clinic Satellite Mission Scheduling with Dynamic Tasking*. Denver, CO, USA: Math. Dept., Univ. Colorado at Denver and Health Sciences Center, 2005.
[29] D. Y. Liao and Y. T. Yang, "Imaging order scheduling of an earth observation satellite," *IEEE Trans. Syst., Man Cybern. C, Appl. Rev.*, vol. 37, no. 5, pp. 794–802, Sep. 2007.
[30] B. Dilkina and B. Havens. (2005). Agile satellite scheduling via permutation search with constraint propagation [Online]. Available: http://www.cs.sfu.ca
[31] J. Wang, X. Zhu, D. Qiu, and L. T. Yang, "Dynamic scheduling for emergency tasks on distributed imaging satellites with task merging," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2275–2285, Jun. 2013.
[32] J. Wang, X. Zhu, J. Zhu, M. Ma, and L. T. Yang. "DMTRH: A real-time scheduling algorithm for multiple earth observation satellites," in *Proc. 14th IEEE Int. Conf. High Perform. Comput. Commun.*, 2012, pp. 673–680.
[33] J. Wang, X. Zhu, J. Zhu, and M. Ma, "Emergency scheduling of multiple imaging satellites with dynamic merging," in *Proc. 12th Int. Conf. Space Oper.*, 2012, pp. 1–12.
[34] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A new fault-tolerant scheduling technique for real-time multiprocessor systems," in *Proc. 2nd Int. Workshop Real-Time Comput. Syst. Appl.*, 1995, pp. 197–202.
[35] S. Ghosh, R. Melhem, and D. Mossé, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 3, pp. 272–284, Mar. 1997.
[36] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 11, pp. 1137–1152, Nov. 1998.
[37] R. Al-Omari, A. K. Somani, and G. Manimaran, "Efficent overloading technique for primary-backup scheduling in real-time systems," *J. Parallel Distrib. Comput.*, vol. 64, no. 5, pp. 629–648, 2004.
[38] C. H. Yang, G. Deconinec, and W. H. Gui, "Fault-tolerant scheduling for real-time embedded control systems," *J. Comput. Sci. Technol.*, vol. 19, no. 2, pp. 191–202, 2004.
[39] 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. Syst. Softw.*, vol. 84, no. 10, pp. 1708–1716, 2011.
[40] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *J. Parallel Comput.*, vol. 32, no. 5, pp. 331–356, 2006.
[41] R. H. Cohen, "Automated spacecraft scheduling-the aster example," in *Proc. Ground Syst. Archit. Workshop*, 2002.
[42] J. C. Pemberton and L. G. Greenwald, "On the need for dynamic scheduling of imaging satellites," *Int. Archives Photogrammetry Remote Sensing Spatial Inf. Sci.*, vol. 34, no. 1, pp. 165–171, 2002.
[43] M. A. A. Mansour and M. M. Dessouky, "A genetic algorithm approach for solving the daily photograph selection problem of the spot5 satellite," *Comput. Ind. Eng.*, vol. 58, no. 3, pp. 509–520, 2010.
[44] S. M. Han, S. W. Beak, K. R. Cho, D. W. Lee, and H. D. Kim, "Satellite mission scheduling using genetic algorithm," in *Proc. SICE Annu. Conf.*, 2008, pp. 1226–1230.

Xiaomin Zhu (M'10) received the BS and MS degrees in computer science from Liaoning Technical University, Liaoning, China, in 2001 and 2004, respectively, and the PhD degree in computer science from Fudan University, Shanghai, China, in 2009. In the same year, he received the Shanghai Excellent Graduate. He is currently an associate professor in the College of Information Systems and Management at National University of Defense Technology, Changsha, China. His research interests include scheduling and resource management in green computing, cluster computing, cloud computing, and multiple satellites. He has published more than 60 research articles in refereed journals and conference proceedings such as *IEEE TC*, *IEEE TPDS*, *JPDC*, *JSS*, *ICPP*, *CCGrid*, and so on. He is an editorial board member of the *Journal of Big Data and Information Analytics*. He is a member of the IEEE, the IEEE Communication Society, and the ACM.

Jianjiang Wang received the BS degree in information systems from National University of Defense Technology, China, in 2009. He is currently working toward the PhD degree in the School of Information System and Management at National University of Defense Technology. He is currently in Research Center for Operations Mangement, Faculty of Economics and Bussiness, KU Leuven, for visiting study. His research interests include satellite scheduling, multiobjective combinational optimization, and uncertain programming.

Xiao Qin (S'99-M'04-SM'09) received the BS and MS degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. He is currently an associate professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He received US National Science Foundation (NSF) CAREER Award in 2009. His research is supported by the NSF, Auburn University, and Intel Corporation. He has been on the program committees of various international conferences, including IEEE Cluster, IEEE MSST, IEEE IPCCC, and ICPP. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He is a member of the ACM and a senior member of the IEEE.



Ji Wang received the BS degree in information systems from National University of Defense Technology, China, in 2008. He is currently working toward the MS degree in the School of Information System and Management at National University of Defense Technology. His research interests include real-time systems, fault-tolerance, and cloud computing.

Zhong Liu received the BS degree in physics from Central China Normal University, Wuhan, China, in 1990, and the MSc degree in computer science and the PhD degree in management science from National University of Defense Technology, Changsha, China, in 1997 and 2000, respectively. He is currently a professor with the College of Information Systems and Management, National University of Defense Technology. His research interests include information management and decision-making support technology. He is a member of the IEEE.

Erik Demeulemeester received the degree in commercial engineer (field of Management Informatics) in 1987, the degree of master's of business administration in 1988, and the PhD degree in 1992, all from the KU Leuven. The title of the PhD degree was Optimal algorithms for various classes of multiple resource-constrained project scheduling problems. He is a professor in the Research Center for Operations Management at the KU Leuven. He is currently a full professor and teaches a course on quality management, a doctoral course, entitled Combinatorial optimization and local search techniques as well as a seminar on production and logistics. His main research interests are situated in the field of project scheduling, personnel scheduling and health care planning and he has published many papers on these topics. He is a member of the editorial board of both the *Journal of Scheduling and Computers* and *Operations Research*.