Study of failure detection and recovery in manufacturing grid resource service scheduling

Fei Tao\textsuperscript{a}\textsuperscript{b}; Yefa Hu\textsuperscript{a}; Dongming Zhao\textsuperscript{c}; Zude Zhou\textsuperscript{c}

\textsuperscript{a} School of Mechanical and Electronic Engineering, Wuhan University of Technology, Wuhan, PR China
\textsuperscript{b} Hubei Digital Manufacturing Key Laboratory, Wuhan University of Technology, Wuhan, PR China
\textsuperscript{c} Department of Electrical and Computer Engineering, The University of Michigan, Dearborn, MI, USA

First published on: 31 October 2008

To cite this Article Tao, Fei, Hu, Yefa, Zhao, Dongming and Zhou, Zude(2010) 'Study of failure detection and recovery in manufacturing grid resource service scheduling', International Journal of Production Research, 48: 1, 69 — 94, First published on: 31 October 2008 (iFirst)

To link to this Article: DOI: 10.1080/00207540802275871
URL: http://dx.doi.org/10.1080/00207540802275871
Study of failure detection and recovery in manufacturing grid resource service scheduling

Fei Tao\textsuperscript{ab}\textsuperscript{*}, Yefa Hu\textsuperscript{b}, Dongming Zhao\textsuperscript{c} and Zude Zhou\textsuperscript{b}

\textsuperscript{a}School of Mechanical and Electronic Engineering, Wuhan University of Technology, Wuhan, PR China; \textsuperscript{b}Hubei Digital Manufacturing Key Laboratory, Wuhan University of Technology, Wuhan, PR China; \textsuperscript{c}Department of Electrical and Computer Engineering, The University of Michigan, Dearborn, MI, USA

\textit{(Received 3 December 2007; final version received 4 June 2008)}

A Manufacturing Grid (MGrid) system is different from a conventional distributed computing system due to its focus on larger-scale, distributed and heterogeneous manufacturing resource and service sharing, including equipment resources, application system resources, material resources, technical resources, public service resources, etc., where remote resource service scheduling (e.g., remote accessing, information communication) has a large influence on the MGrid quality of service (QoS). The probability of failure is higher in MGrid resource service scheduling and failures affect task execution and the quality of service of the MGrid fatally. Therefore, this paper focuses on failures in the MGrid resource service scheduling process. The potential failures that can occur during MGrid resource service scheduling are investigated. Thirteen failures are first defined in detail and are classified into four categories: (a) virtual-link-related failures, (b) resource-service-related failures, (c) task-related failures, and (d) application-related failures. A failure management system of the MGrid system is presented associated with its architecture. Corresponding detection mechanisms and methods for each defined failure are presented in detail, as well as the corresponding failure recovery methods. The implementation and simulation results indicate that our approaches are sound for promoting a successful scheduling rate and shortening the total execution time of the MGrid resource service.

\textbf{Keywords:} manufacturing grid (MGrid); failure; resource service; failure detection; failure recovery

1. Introduction

A manufacturing grid (MGrid) utilises grid technologies, information technologies, computer and advanced management technologies, etc., to overcome the barrier resulting from the spatial distance between collaborating corporations, and makes a connection between distributed and heterogeneous manufacturing resources, including design resources, equipment resources, human resources, application system resources, etc. In an MGrid, various manufacturing resources distributed in heterogeneous systems and multiple sites are encapsulated into different corresponding resource service templates, and
users can use all resources and services in the MGrid as if they were local (Fan et al. 2003, Qiu 2004, Tao et al. 2008).

In an MGrid there are primarily two types of users (Tao et al. 2008): (a) a resource enterprise or a resource service provider (RSP), and (b) a user enterprise or a resource service demander (RSD). The RSP publishes its idle resources, products, manufacturing abilities, and related service capabilities, and provides a manufacturing resource service that meets the requirements of an MGrid. The RSD searches for the optimised manufacturing resource and service according to demand, and selects corresponding partners to establish a virtual collaboration manufacturing net. The process of a resource service transaction between a RSD and a RSP is shown in Figure 1.

In Figure 1, when a RSD and a RSP carry out a resource service transaction, the transaction will be stopped or fail if any of the following phenomena occur.

- The virtual link between two entities (i.e. the RSD and the RSP) is disconnected, or the bandwidth between them decreases suddenly for some reason.
- The resource service experiences failure during a transaction process, e.g. the resource service quits suddenly, is overloaded, or there is a mismatch during resource service composition, etc.
- The submitted task experiences failure during a transaction process, e.g. task cancellation, task suspension, the task requirement cannot be satisfied, the task and resource service are mismatched, etc.
- Wrong access permission, full disk or memory outage, memory leak, numerical exception, etc.

![Figure 1. MGrid resource service transaction between RSD and RSP (Tao et al. 2007).](image-url)
The above phenomena are all defined as failures during MGrid resource service scheduling. In order to enable MGrid resource service scheduling to be performed successfully, the following three issues must be effectively addressed.

(a) What failures will occur during MGrid resource service scheduling?
(b) How does the system learn that a failure has occurred? Namely, failure detection.
(c) How to deal with the detected failures? Namely, failure recovery.

Motivated by addressing the above issues and realising MGrid resource service scheduling, we carried out studies on failure management in MGrid systems, including failure classification and definition, the failure management system and its architecture, failure detection, and failure recovery.

The main contributions of this paper are the following.

(1) A classification and definition of failures in MGrid is presented. MGrid resource service failures are classified into four categories, i.e. virtual-link-related failures, resource-service-related failures, task-related failures, and application-related failures, and 13 failures are defined in detail.
(2) A failure management system is proposed. Corresponding failure detection mechanisms and methods are proposed for each defined failure.
(3) An ECA (event–condition–action)-rules-based failure recovery mechanism is put forward, and the corresponding failure recovery methods are presented.

The paper is organised as follows. In Section 1 we introduce a brief overview of MGrid and the failure problems encountered in MGrid resource service transactions. In Section 2 we investigate related works and discuss the novelty and significance of our research. A definition and classification of possible failures in the MGrid resource service scheduling process are presented in Section 3. We put forward a failure management mechanism for the MGrid system and its architecture in Section 4. Failure detection methods are proposed in Section 5 and ECA rules-based failure recovery methods are proposed in Section 6. Performance simulations of the proposed methods are described in Section 7. The conclusion and future work are proposed in Section 8.

2. Related work

2.1 Failure management in manufacturing systems

Research into MGrids is in its initial stage and current work primarily concentrates on the concept and architecture (Fan et al. 2003, Hu et al. 2006), the application of a prototype platform (Chen et al. 2003, Zheng et al. 2005), the application foreground (Huang et al. 2005), etc. To the best of our knowledge, MGrid resource service failure management, including failure detection and recovery, which is one of the key technologies for realising a dynamic MGrid execution platform, has not been adequately addressed. Studies on failure management in manufacturing systems are addressed below.

Alcaraz-Mejia et al. (2006, 2007) studied fault recovery of discrete manufacturing systems through the reconfiguration of the controller and proposed a technique for fault recovery that exploits the redundancies included in the Petri net model. They assumed that the system is controlled using a Petri net regulation controller scheme and presented
a novel control reconfiguration technique for fault recovery. Their main contribution is an incidence-matrix-based technique and an Abelian group of Petri net modules to re-compute the controller each time that a resource fails. Fries (2007) researched multi-agent fault diagnosis in manufacturing systems using multiple fuzzy intelligent agents that examine the problem domain from a variety of perspectives. Du et al. (2006) discussed the application of a distributed sensor system for fault detection and isolation in multistage manufacturing systems. Fletcher and Deen (2001) researched a model of fault-tolerant holonic manufacturing systems (HMS) where each holon’s activities are controlled by an intelligent software agent. Multiple agents schedule actions, resolve conflicts and manage information to produce, transport, assemble, inspect and store customised products. The model provides robustness and distribution transparency across a shop-floor where unpredictable failures occur with machines, control software and communication networks. Ulieru and Norrie (2000) introduced a novel distributed fault recovery method that could be useful in the design of re-configurable holonic manufacturing systems. It can accomplish fault recovery by the re-distribution of tasks in the case where a resource fails and can also re-configure production by the dynamic re-allocation of resources for newly received high-priority tasks.

Byrne and Sheahan (2005, 2006) used integrated risk minimisation (IRM) methodology to reduce the potential risks of failure in high-volume manufacturing, and the proposed methodology was implemented and tested in the electroplating process. They also researched the technology and methods for automation of the inspection of surface quality by the application of in-line vision systems (i.e. a high-speed grey-scale system, a standard colour system and a high-speed colour system), and applied them to a high-volume connector manufacturer. The overall benefit of the method as reported in their paper was an improvement in the defect rate and a reduction of the risk priority number on the failure, modes and effects analysis of the process.

Mannar and Ceglarek (2004) proposed a machine learning approach using rough sets to provide rapid diagnosis of dimensional failures. In their later work (Mannar et al. 2006), they proposed a novel fault region localisation methodology to link warranty failures to manufacturing measurements (hence to design and process parameters) for diagnosing warranty failures and to perform tolerance revaluation. The methodology consists of identifying the relations between warranty failures and design/process variables using rough-sets-based analysis on training data consisting of warranty information and manufacturing measurements. The methodology expands the rough-set-based analysis by introducing parameters for the inclusion of noise and the uncertainty of warranty data classes. Based on the identified parameters related to the failure, a revaluation of the original tolerances can be performed to improve product robustness.

The failures considered in the above papers primarily result from physical components of the manufacturing system. Although these papers propose an efficient failure tolerance (including both failure detection and recovery) in traditional manufacturing systems, their schemes do not suit SOA (service-oriented architecture) manufacturing systems such as MGrid. In an MGrid system, all resources or manufacturing capabilities are encapsulated into the service, and during the MGrid resource service scheduling process, the system does not consider the physical state (such as temperature or pressure) of the resources. It only considers the availability of the encapsulated services, such as accessibility and load.
2.2 Failure management in distributed systems

In current grid application systems, tools such as Globus (Foster and Kesselman 1997, 1998), Condor-G (Frey et al. 2002), Nimnod-G (Buyya et al. 2000), Ninf-G (Tanaka et al. 2003) and others (Iamnitchi and Foster 2000, Dialani et al. 2002, Hwang and Kesselman 2003) have addressed either failure tolerance issues and have not provided a general mechanism for resolving failures, or different applications have adopted \textit{ad hoc} fault-tolerance mechanisms that cannot be reused, or shared. Furthermore, several related works concerning failure management in a distributed computing system can be found in the literature. Fang et al. (2007) proposed a fault-tolerant web service SOAP (Simple Object Access Protocol) called fault-tolerant SOAP (or FT-SOAP) by which web services can be built with a higher resilience to failure. Luckow and Schnor (2008) presented a fault-tolerant service framework for an MPI application in a grid system, namely Migol. Domingues et al. (2007) investigated sabotage-tolerance and trust management in desktop grid computing, and presented a mechanism to detect sabotage attempts based on a checkpoint comparison. Lee et al. (2005) studied the fault-tolerance service in grid computing, and classified failures into process failures, processor failures, and network failures, and a fault detector and fault manager components were proposed. Khanli and Analoui (2008) researched grid resource fault management based on ECA (event–condition–action) rules. Huedo et al. (2006) investigated the evaluation method for the reliability of a computational grid from the end-user’s point of view, and proposed six kinds of failure.

The resources considered in the above research are primarily computational resources, such as processors, memory, storage, networks and processes, which are inadequate in an MGrid system. In addition, the detailed fault detection implementation methods have not been studied, and they do not provide a detailed and executable mechanism or method for handling the detected failures.

Therefore, in this paper, based on the above work, we investigate all potential failures during MGrid resource service scheduling, and define four categories and 13 failures in detail. These are: (a) virtual-link-related failures; (b) resource-service-related failures; (c) task-related failures; and (d) application-related failures. We propose a failure management system to provide a failure tolerance service. Detailed detection mechanisms and methods for each failure are put forward, as well as the methods for detecting each failure.

3. Definition and classification of an MGrid failure

It is a failure if and only if one of the following two conditions is satisfied (but not both) (Lee et al. 2005):

- a resource service stops due to a resource crash; or
- the availability of a resource does not meet the minimum level of a task.

As discussed in the Introduction, all the factors, such as the virtual link, resource service, task, and application involved in MGrid resource service scheduling, can cause a failure. Therefore, we investigate and define the possible failures from the above four aspects, namely:

- virtual-link-related failure,
- resource-service-related failure,
- task-related failure, and
- application-related failure.
3.1 Virtual-link-related failure
The virtual link (VL) is a general connection between two entities (e.g., resource service provider (RSP) and resource service demander (RSD), resource and resource, user and user) in an MGrid. A VL can consist of various elements, such as routers, computers, internet/intranet, wireless communication channels, etc. In this paper, we primarily consider the following two failures resulting from a virtual link.

- **Virtual link disconnected failure (VL_Disconnect_Failure).** A VL_Disconnect_Failure is defined as the failure caused by a disconnected virtual link. The connection of a virtual link could unpredictably fail due to the power, hardware, the base software, etc. Without the connection, any service transaction in MGird has to be stopped.
- **Inadequate bandwidth failure (Bandwidth_Failure).** MGrid is a Web-based manufacturing mode, so the network bandwidth is critical to the realisation of an MGrid due to the network traffic varying dynamically. We define the failure that causes an invalid resource or task connection, or data transmission and accessing due to inadequate bandwidth in the MGrid resource service scheduling process, as a Bandwidth_Failure.

3.2 Resource-service-related failure
In the MGrid resource service scheduling process, resource services are the scheduling objects and the executors of tasks. Therefore, a change in the resource service state, such as disconnection from the MGrid system, overloading or a decreasing QoS, would result in a failure of MGrid resource service scheduling. There are many failures caused by the resource service, and taking into account the importance and necessity of MGrid resource service scheduling we primarily consider the following four kinds of failure resulting from resource services.

- **Resource service quit failure (RS_Quit_Failure).** In an MGrid, a RSP can add or move a resource service freely at any time; moreover, a resource service can be shut down by system administrators freely for certain reasons, which will result in the executing task (or the task allocated to the resource service) not being completed in time. Therefore, these failures caused by the resource service being shut down are defined as an RS_Quit_Failure.
- **Resource service overload or saturation failure (RS_Overload_Failure).** Resource services are shared by all users of an MGrid; hence, the resource load changes dynamically. We only consider the failure resulting from resource overload if it results in a task not being completed on time according to its quality requirements, and define these failures as an RS_Overload_Failure.
- **Resource service composition failure (RS_Composition_Failure).** Due to the complexity of manufacturing processes, a submitted manufacturing task (or resource service request) in an MGrid usually cannot be completed by invoking only one resource service. It is usually decomposed into several subtasks, and completed by invoking several resource services in sequence, which is defined as the resource service composition (Tao 2008). Assume that $RS_i$ and $RS_j$ are two arbitrary neighbouring resource services in a composition path, and $RS_j$ is the direct successor (Zeng et al. 2004) of $RS_i$. During the MGrid resource services
composition process, the following four kinds of failures would be generated between \( RS_i \) and \( RS_j \).

(I) **Conceptual mismatch between \( RS_i \) and \( RS_j \).** For example,
- \( RS_i \) provides a ‘cutting’ service, but \( RS_j \) requires a ‘planning’ service, so there is a gap between \( RS_i \) and \( RS_j \). We define this kind of concept relationship as \( RS_i \perp RS_j \).
- \( RS_i \) is a superclass of \( RS_j \). We define this kind of concept relationship as \( RS_i \supseteq RS_j \).
- \( RS_i \) is a subclass of \( RS_j \). We define this kind of concept relationship as \( RS_i \subseteq RS_j \).
- \( RS_i \) is a part of \( RS_j \). We define this kind of concept relationship as \( RS_i \in RS_j \).

(II) **Data mismatch between \( RS_i \) and \( RS_j \).** For example,
- \( RS_i \) and \( RS_j \) have the same data type associated with a parameter (such as input parameter, output parameter, QoS parameter), but they have different units, e.g. \( RS_i \) uses feet, but \( RS_j \) requires meters. We define this kind of relationship as \( RS_i \approx RS_j \).
- \( RS_i \) and \( RS_j \) have the same parameter concepts, but different data types, e.g. \( RS_i \) employs a ‘string’ data type to define a parameter, but \( RS_j \) requires a ‘character’ data type for the corresponding parameters. We define this kind of relationship between \( RS_i \) and \( RS_j \) as \( RS_i \neq RS_j \).

(III) **Attributes mismatch between \( RS_i \) and \( RS_j \).** \( RS_j \) has more attribute requirements than \( RS_i \) can offer. For example, \( RS_i \) requires five attributes, but \( RS_j \) only has two. We define this kind of relationship between \( RS_i \) and \( RS_j \) as \( RS_i < RS_j \).

(IV) **QoS inconsistent.** In resource service composition, the QoS consistency between two neighbouring resource services must be taken into account. If the consistency between two neighbouring resource services is bad, it would result in the failure of the whole composition.

(6) Resource service ability change failure (\( RS\_AbilityChange\_Failure \)). During execution, a resource service’s ability may change suddenly for several reasons, such as hardware failure, etc., which would cause the task it served to be moved or rescheduled. We define the failure that results in the task allocated not being accomplished successfully due to the change in the resource service as an \( RS\_AbilityChange\_Failure \).

### 3.3 Task-related failure
A task is the executed object in an MGrid, and a task can be canceled, suspended, or mismatched with a resource service for some reason, which will result in failure of MGrid resource service scheduling. There are many failures caused by a task, and taking into account the importance and necessity of MGrid resource service scheduling, we primarily consider the following four kinds of failures resulting from tasks.

(7) Task cancellation failure (\( Task\_Cancel\_Failure \)). A submitted or executing task can be cancelled for various reasons, e.g. stopped by the user when exceeding the wall
time limit, or by the system administrator to preserve system performance, etc. We define the failure in MGrid resource service scheduling caused by task cancellation as Task_Cancel_Failure.

(8) Task suspension failure (Task_Suspension_Failure). A task can be suspended for various reasons. For example, (a) the requirements of the task are too high to be satisfied at the current time and are out of negotiation; (b) higher-priority tasks continuously entering the system could prevent the execution of the task; and (c) the system administrator could freely hold a waiting task for system performance, etc.

(9) Mismatch failure between a task and the resource service (Task_RS_Mismatch_Failure). If an assigned resource service cannot satisfy the requirements of a task for some reason (e.g., mismatch, the ability of the resource service to change suddenly, etc.), then the system has to search for and assign a new resource service for the task, or the task has to be moved. Failures caused by a mismatch between a task and the resource service during MGrid resource service scheduling are defined as Task_RS_Mismatch_Failure.

(10) Task requirements change failure (Task_RequireChange_Failure). The requirements of a task could be changed by the user. If the requirements of the task are changed to a higher level, such as higher precision, shorter execution time, lower price, etc., this will cause a rescheduling of the task. Failures resulting from changes in task requirements are defined as Task_RequireChange_Failure.

3.4 Application-related failure

(11) Trust failure (Trust Failure). In an MGrid, there are mainly two kinds of users (Tao et al. 2007): (a) a resource enterprise or a resource service provider (RSP), and (b) a user enterprise or a resource service demander (RSD). The RSP publishes its idle resource service to meet the users’ requirements in the MGrid. The RSD searches for the optimal manufacturing resource and service it requires, and selects the corresponding partner to establish the virtual collaboration manufacturing net. When the RSD and the RSP have a resource service transaction with each other, the following problems will be generated.

From the viewpoint of the RSD, before submitting its manufacturing task, there are three problems (Tao et al. 2007).

- The RSD must know whether the RSP (to be visited), who provides the required resource service, is trustworthy or has the ability to faithfully accomplish the task.
- If the answer to the first question is yes, then the RSD must determine if the RSP will trust in it and be willing to accept and execute the submitted manufacturing task.
- The RSD must confirm that the RSP will not destroy the submitted manufacturing task, and the latter will not provide unsuitable results and other uncertainties.

From the viewpoint of the RSP, before the submitted manufacturing task is executed, there are two problems.

- The RSP must check the validity of the manufacturing task submitted by the RSD (to be served), and make sure the RSD will not submit unsuitable or destructive tasks that would destroy the MGrid system.
• The RSP must be certain that the task was not destroyed and tampered with by third parties before submission.

Therefore, undesired actions or results during the resource service transaction resulting from trust problems between RSDs and RSPs will reduce the QoS of the MGrid, such as the quality of resource scheduling, the quality of resources management, etc.

(12) Application design or coding failures (App_DesignCode_Failure), such as memory leaks, numerical exceptions, etc.

(13) Accessing right failures (App_AccessRight_Failure), such as wrong access permission, full disk or memory outages.

4. Architecture of the MGrid failure management system

Figure 2 shows the architecture of the failure management system (FMS) in MGrid. The proposed FMS can provide a failure detection service and failure recovery service during MGrid resource service scheduling. The main components of the proposed FMS

---

The text above is a continuation of the discussion on the failure management system in MGrid. It explains the need for the RSP to ensure that tasks are not destroyed or tampered with by third parties before submission. It also discusses how trust issues between RSDs and RSPs can affect the QoS of the MGrid, leading to issues such as resource scheduling and management. The text further lists two types of failures: application design or coding failures and accessing right failures.

The section then introduces the architecture of the MGrid failure management system (FMS). It explains how the FMS can provide failure detection and recovery services during resource service scheduling. The main components of the proposed FMS are discussed, including the moving of tasks, re-scheduling, and the evaluation of the system's performance.

Figure 2 illustrates the architecture of the FMS in MGrid, showing the relationship between the different components and how they work together to manage failures. The figure includes labels such as MGrid Resource Service, Optimal-Selection System, Information Service Model, and Failure Recovery System, among others, which are crucial for understanding the FMS's operation.

---

The diagram in Figure 2 visually represents the architecture of the failure management system in MGrid. It illustrates how different components interact to manage failures, including task migration, re-scheduling, and failure detection and recovery. The labels on the diagram correspond to the text, providing a clear visual representation of the FMS's components and their functions.
include an information service model (ISM), an MGrid resource service optimal-selection system (MGRSOSS), a failure detector (FD) and failure recovery (FR).

(1) MGrid resource service optimal-selection system (MGRSOSS). MGRSOSS is the source of failure generation. As presented in our previous work (Tao et al. 2008), the key technology to realise MGrid resource service scheduling is resource service optimal-selection, including the operations of resource service math and search, resource service QoS synthetic processing (QoS extraction, QoS evaluation, QoS comparison), resource service composition, etc. Almost all failures are generated in these operations. The detailed model, workflow, and algorithms of resource service optimal-selection in MGrid have been presented in the authors’ earlier research (Tao et al. 2008).

(2) Information service model (ISM). The ISM provides FD, FR, and MGRSOSS with data and information. It consists of the GIIS (Grid Information Index Service) and the GRIS (Grid Resource Information Service), provided by Globus (Foster et al. 2001, Du et al. 2002), nine kinds of resource service encapsulation templates developed by the authors (Tao et al. 2008), and the QoS database for depositing QoS information and data.

(3) Failure detector (FD). This monitors the state of the virtual link, the resource service, the task, and the application involved in MGrid resource service optimal allocation. It consists of the following. (a) Four monitors, i.e. the VL_Monitor, RS_Monitor, Task_Monitor, and App_Monitor, which are responsible for monitoring the state of the virtual link, the resource service, the task, and the application involved in the MGrid resource service optimal-allocation process. (b) The failure decision agent, which decides on the occurrence of a failure by analysing information states of the virtual link, the resource service, the task, and the application; it identifies related failures such as a VL-related failure, a RS-related failure, a task-related failure, or an application-related failure; if a failure occurs, it reports the failure information to the failure recovery model. (c) The communication agent, which manages the communication between detectors and agents.

(4) Failure recovery (FR). This is responsible for analysing the detected failure, and provides a failure recovery service. It consists of the following. (a) An ECA-rule-based failure recovery service, which provides a failure recovery service based on ECA rules. Detailed methods of ECA-rule-based failure recovery are presented in Section 6. (b) A communication agent, which manages the communication between detectors and agents. (c) A rescheduling evaluation agent. If the detected failure cannot be recovered based on ECA rules, then the rescheduling evaluation agent evaluates the total benefit the system can obtain from task migration and decides whether a task migration occurs or not. (d) A task migration agent. If it receives a rescheduling result from the rescheduling evaluation agent, it requests the reallocation of a resource service to the task or restarts the task.

5. Detection of an MGrid failure

5.1 Detection of virtual-link-related failures

- Detection of a VL_Disconnect_Failure. A VL_Disconnect_Failure can usually be detected by the security policy or system middleware embedded in an MGrid. In this paper, we employ the Globus GRAM (Grid Resource Allocation and
Management) service, i.e. GRAM callbacks, to detect a virtual link disconnection failure (Czajkowski et al. 1998). GRAM callbacks notify submission failures, which include connection, authentication and authorisation, RSL (Resource Specification Language) parsing, executable or input staging (Huedo et al. 2006).

- **Detection of Bandwidth_Failure.** For Bandwidth_Failure, we employ two criteria, 
communication time and probability of successful communication, to judge whether a Bandwidth_Failure has occurred between two entities. Assume that the virtual link between entities $A$ and $B$ is denoted $VL(A, B)$, the total information exchanged through $VL(A, B)$ is $SumInfor(A, B)$, and the speed (bandwidth) of $VL(A, B)$ is $V(A, B)$, then the estimated communication time, $T_c(A, B)$, between $A$ and $B$ can be calculated as follows:

$$T_c(A, B) = \frac{SumInfor(A, B)}{V(A, B)} + Waite(A, B),$$  \hspace{1cm} (1)

where $Waite(A, B)$ is the waiting time. Assume that the failure intensity of $VL(A, B)$ is $\alpha(A, B)$, and the failure intensities of entities $A$ and $B$ are $\alpha(A)$ and $\alpha(B)$, respectively. Therefore, the probability of successful communication, $S_c(A, B)$, between $A$ and $B$ can be expressed as

$$S_c(A, B) = \exp\{-(\alpha(A) + \alpha(B) + \alpha(A, B))T_c(A, B))\}. \hspace{1cm} (2)$$

Set the limit of requested communication time between $A$ and $B$ as $T_c(A, B)$, and the lowest requested probability of successful communication as $S_c(A, B)$. If $T_c(A, B) > T_c(A, B)$ or $S_c(A, B) < S_c(A, B)$, then the system determines that a Bandwidth_Failure has occurred.

### 5.2 Detection of resource-service-related failures

- **Detection of an RS_Quit_Failure.** In order to detect an RS_Quit_Failure, the resource service monitor (i.e. RS_Monitor in Figure 2) checks, on a regular basis, that each resource service associated with a task is alive. If the resource service fails to respond to the RS_Monitor, then the system determines that the resource service has failed or quit, i.e. an RS_Quit_Failure has occurred. The main advantage of this detection mechanism is that, as the RS_Monitor periodically polls resource services in MGrid, resource services may be found to be faulty prior to the moment when an application or a task would want to use them; therefore, they can be replaced with a zero replacement waiting time.

- **Detection of an RS_Overload_Failure.** Let $RS_i$ be an arbitrary resource service that is modeled as a vector with four parameters, $RS_i = (B_i, M_i, \Gamma_i, D_i)$, where $B_i$ is the bandwidth of the storage subsystem of $RS_i$, $M_i$ is the memory capacity, $\Gamma_i$ is the set of tasks allocated to $RS_i$ (assume that there are $k$ tasks allocated to $RS_i$, then $\Gamma_i = \{Task_1, Task_2, \ldots, Task_j, \ldots, Task_k\}$), and $D_i$ is an array of datasets accessed and generated by an application.

An arbitrary task is expressed as $Task_j = (n_{ij}^{RS}, d_{ij}, et_{ij}, td_{ij}, m_j)$, where $n_{ij}^{RS}$ is the number of requested resource services, $d_{ij}$ is the total information that has to be exchanged, $et_{ij}$ is the required execution time, $td_{ij}$ is the time spent on data communication ($td_{ij} = \sum_{j=1}^{n} td_{ij}$ and $td_{ij} = d_{ij}/V(i,j)$), where $d_{ij}$ is the amount of accessed data and $V(i,j)$ is the bandwidth of the $i$th dedicated storage subsystem (i.e. $RS_i$), and $m_j$ is the memory requirements of $Task_j$. 

We make use of the following three equations to quantify the communication, computational and data accessing load of $RS_i$:

$$\text{Communication}_{RS_i} = \sum_{\text{Task}_{j} \in \Gamma_i} \left( \frac{d_{i,j}}{V(i,j)} \right),$$

$$\text{Computational}_{RS_i} = \sum_{\text{Task}_{j} \in \Gamma_i} (n_j^{RS} \cdot et_j),$$

$$\text{Data}_{RS_i} = \sum_{\text{Task}_{j} \in \Gamma_i} d_{i,j}.$$  \hfill (3)

The limits of the communication time, execution time and data capability of $RS_i$ are $\text{Lim}^{CT}_{RS_i}$, $\text{Lim}^{ET}_{RS_i}$, and $\text{Lim}^{DC}_{RS_i}$, respectively. Therefore, if any of the following three conditions is satisfied:

$$\text{Computational}_{RS_i} = \sum_{\text{Task}_{j} \in \Gamma_i} (n_j^{RS} \cdot et_j) > \text{Lim}^{ET}_{RS_i},$$

$$\text{Communication}_{RS_i} = \sum_{\text{Task}_{j} \in \Gamma_i} \left( \frac{d_{i,j}}{V(i,j)} \right) > \text{Lim}^{CT}_{RS_i},$$

$$\text{Data}_{RS_i} = \sum_{\text{Task}_{j} \in \Gamma_i} d_{i,j} > \text{Lim}^{DC}_{RS_i},$$

the system determines that an $\text{RS\_Overload\_Failure}$ has occurred.

- **Detection of an RS\_Composition\_Failure.** In an MGrid, $RS_i$ and $RS_j$ are annotated with OWL-S, and match and search mechanisms (Tao 2008) are used in order to identify a list of possible Web services. Identification rules for the above RS\_Composition\_Failure are as follows.

  (I) **Conceptual mismatch detection rules**

  - If $RS_i \subset RS_k \&\& RS_k \not\subset RS_j$, then the system determines that a conceptual mismatch (concept gap) has occurred between $RS_i$ and $RS_j$.
  - If $RS_i \subset RS_k \&\& RS_k \not\subset RS_j$, then the system determines that a conceptual mismatch ($RS_j$ is a superclass of $RS_i$, $RS_i$ cannot satisfy $RS_j$’s requirements) has occurred between $RS_i$ and $RS_j$.

  (II) **Data mismatch detection rules**

  - If $\text{DUnitTransfer}(RS_i) = RS_j$, then $RS_i \approx RS_j$, and the system determines that a data mismatch (same data type but different units) has occurred. $\text{DUnitTransfer}()$ is a data unit transformation function.
  - If $\text{DTypeTransfer}(RS_i) = RS_j$, then $RS_i \neq RS_j$, and the system determines that a data mismatch (same concept but different data type) has occurred. $\text{DTypeTransfer}()$ is a data-type transformation function.

  (III) **Attribute mismatch detection rules**

  - If $RS_i < RS_j \&\& RS_i \cap \mathcal{A}(RS_j) \neq \emptyset$, it demonstrates that $RS_i$’s attributes cannot satisfy $RS_j$’s requirements, and the system determines that an attributes\_Failure has occurred. $\mathcal{A}()$ is a split function.
The above detection rules for an RS_Composition_Failure are examples, but not an exhaustive list.

(IV) QoS inconsistent detection rules. Assume that $\text{Dim}(Q_{\text{inputs}}^i)$ and $\text{Dim}(Q_{\text{outputs}}^j)$ are the number of parameters of $RS_i$ and $RS_j$, respectively, for $\forall 1 \leq m \leq \text{Dim}(Q_{\text{inputs}}^i)$, $\exists 1 \leq n \leq \text{Dim}(Q_{\text{outputs}}^j)$, if $RS_i$ and $RS_j$ are subject to

\[
\begin{align*}
    &d_{j,m} \leq d_{i,n} \quad (d_{j,m} \text{ is a single and negative parameter}), \\
    &d_{j,m} \geq d_{i,n} \quad (d_{j,m} \text{ is a single and positive parameter}), \\
    &d_{j,m} \leq d_{i,n} \quad (d_{j,m} \text{ is an interval number parameter}).
\end{align*}
\]

(9)

It demonstrates that $RS_i$ and $RS_j$ are QoS consistent, and the composition path they have located is effective. Otherwise, the system determines that an RS_Composition_Failure has occurred between $RS_i$ and $RS_j$.

5.3 Detection of task-related failures

- Detection of a Task_Cancel_Failure and a Task_Suspension_Failure. The task exit conditions can be checked through the drama_wif*() routines included in the DRMAA interface (Rajic et al. 2003), which is fully implemented in GridWay (Huedo et al. 2004), and therefore the appropriate corrective action can be undertaken if application-related failures occur.

- Detection of a Task_Suspension_Failure. In order to detect a Task_Suspension_Failure, the Task_Monitor in Figure 2 monitors local tasks, and checks, on a regular basis, the state of each task. A Task_Suspension_Failure is detected when a task remains in its SUSPEND state longer than the given threshold specified by the user.

- Detection of a Task_RS_Mismatch_Failure. Assume that $Task_j$ is the requirement description of a subtask or a task that cannot be decomposed (i.e. the requested resource service), and that $RS_i$ is a candidate resource service for $Task_j$. The matching model between $Task_j$ and $RS_i$ is defined as follows:

\[
\text{Match}(RS_i, Task_j) = \prod (\text{Match}_{\text{bas}}(RS_i, Task_j), \text{Match}_{\text{foo}}(RS_i, Task_j), \text{Match}_{\text{QoS}}(RS_i, Task_j)),
\]

(10)

where

- $\text{Match}_{\text{bas}}(RS_i, Task_j)$ is the basic matching function between $Task_j$ and $RS_i$. It is primarily responsible for matching the general information of $RS_i$ and $Task_j$, such as ServiceName and ServiceDescription

\[
\text{Match}_{\text{bas}}(RS_i, Task_j) = \left( \omega_1 \times \text{Match}_w(RS_i\cdot\text{ServiceName}, Task_j\cdot\text{ServiceName}) + \omega_2 \times \text{Match}_T(RS_i\cdot\text{ServiceDescription}, Task_j\cdot\text{ServiceDescription}) \right),
\]

(11)

where $\omega_1$ and $\omega_2$ are the weights of ServiceName and ServiceDescription, respectively, and $0 \leq \omega_1, \omega_2 \leq 1$ and $\omega_1 + \omega_2 = 1$; $\text{Match}_w()$ is the similarity matching function of word concept describing information, and $\text{Match}_T()$ is the
similarity matching function of sentence describing information developed in our previous work (Tao 2008).

- Match$_{i/o}(RS_i, Task_j)$ is the I/O (input/output) parameter matching function between Task$_j$ and RS$_i$. It is primarily responsible for matching the I/O parameter describing information of RS$_i$ and Task$_j$

\[
\text{Match}_{i/o}(RS_i, Task_j) = \left( \sum_{i=1}^{n_i} \omega_d_i \text{Match}_D(RS_i; d_i, RS_j; d_i) + \sum_{i=1}^{n_c} \omega_c_i \text{Match}_C(RS_i; c_i, Task_j; c_i) + \sum_{i=1}^{n_w} \omega_w_i \text{Match}_W(RS_i; w_i, Task_j; w_i) \right)
\]

where $d_i (d_i \in D)$, $c_i (c_i \in C)$, and $w_i (w_i \in W)$ are the $i$th, $i_2$th, and $i_3$th parameters in $D$, $C$, and $W$, respectively; $D$, $C$, $W$ are the sets of number parameters, word parameters, and entity classes parameters, respectively; $\omega_d_i$, $\omega_c_i$, $\omega_w_i$ are the corresponding weights of $d_i$, $c_i$, and $w_i$, respectively; $0 \leq \omega_d_i, \omega_c_i, \omega_w_i \leq 1$ and $\sum_{i=1}^{n_i} \omega_d_i = \sum_{i=1}^{n_c} \omega_c_i = \sum_{i=1}^{n_w} \omega_w_i = 1$; Match$_D()$ is the similarity matching function of number describing information and Match$_C()$ is the similarity matching function of entity class describing information developed in our early work (Tao 2008).

- Match$_{QoS}(RS_i, Task_j)$ is the QoS parameter matching function between Task$_j$ and RS$_i$. It is primarily responsible for matching the functional QoS parameter information between Task$_j$ and RS$_i$

\[
\text{Match}_{QoS}(RS_i, Task_j) = \sum_{i=1}^{m} \omega_{QoS_i} \text{Match}_Q(RS_i; QoS_i, Task_j; QoS_i)
\]

where $RS_i$.QoS$_i$ and Task$_j$.QoS$_i$ are the $i$th QoS parameter of RS$_i$ and Task$_j$, respectively, $\omega_{QoS_i}$ is the weight of each corresponding QoS parameter, $0 \leq \omega_{QoS_i} \leq 1$ and $\sum_{i=1}^{m} \omega_{QoS_i} = 1$, and $m$ is the total number of QoS parameters.

- $\prod ()$ is the integrated match processing function. It integrates the matching results of Match$_{bas}(RS_i, Task_j)$, Match$_{QoS}(RS_i, Task_j)$, Match$_{i/o}(RS_i, Task_j)$, and generates an integrated matching result.

Assume that the declared threshold values are $\xi_{bas}$ (i.e. the basic matching threshold), $\xi_{i/o}$ (i.e. the I/O matching threshold), $\xi_{QoS}$ (i.e. the QoS matching threshold) and $\xi$ (i.e. the total matching threshold). Therefore:

(I) If Match$_{bas}(RS_i, Task_j) < \xi_{bas}$, then the system determines that a basic mismatch failure has occurred between RS$_i$ and Task$_j$.

(II) If Match$_{i/o}(RS_i, Task_j) < \xi_{i/o}$, then the system determines that an I/O mismatch failure has occurred between RS$_i$ and Task$_j$.

(III) If Match$_{QoS}(RS_i, Task_j) < \xi_{QoS}$, then the system determines that a QoS mismatch failure has occurred between RS$_i$ and Task$_j$.

(IV) If Match$(RS_i, Task_j) < \xi$, then the system determines that an integrated mismatch failure has occurred between RS$_i$ and Task$_j$.

RS\_AbilityChange\_Failure and Task\_RequireChange\_Failure detection methods are the same as that of a Task\_Resource\_Mismatch\_Failure.
5.4 Detection of application-related failures

- **Detection of a Trust_Failure.** Trust is the belief or faith that a trusting entity (e.g., RSD) has in a trusted entity’s ability (e.g., RSP) in a given context, at a time slot, which gives it confidence to carry out a transaction with the trusted entity. Trust is realised by the concept of a trust-QoS relationship (Tao et al. 2007). Each trust-QoS relationship denotes a trust value from a trusting entity to a trusted entity, for a particular service content and time slot. The trust-QoS relationship is a composite of (1) transaction entities, A, including both the trusting entity and the trusted entity; (2) manufacturing resource service contexts C; (3) trust type Γ, including intra-domain trust-QoS and inter-domain trust-QoS; (4) trust-QoS request type R, including strong-trust request, weak-trust request, and no-trust quest; (5) trust-QoS value E; (6) trust-QoS benefit B; and (7) time slot T. Therefore, the trust-QoS relationship model is represented as a tuple: \((A, C, Γ, R, E, B, T)\) (Tao et al. 2007).

Provided that \(x\) and \(y\) stand for two entities that have a resource service transaction in an MGrid, and \(x\) is the trusting entity and \(y\) is the trusted entity, according to our previous research (Tao et al. 2007) the trust value between \(x\) and \(y\), \(T_{x→y}\), can be evaluated as follows.

(I) If \(x\) and \(y\) belong to the same resource domain, then

\[
T_{x→y} = \alpha \times d_{id} + \beta \times r_{id},
\]

where \(d_{id}\) is the direct trust-QoS evaluation value between \(x\) and \(y\), \(r_{id}\) is the comprehensive combined recommended trust-QoS evaluation value of each recommender’s individual recommended trust with \(y\), and \(\alpha, \beta\) are the weights of \(d_{id}\) and \(r_{id}\), respectively.

(II) If \(x\) and \(y\) belong to two different resource domains \(A\) and \(B\), respectively, then

\[
T^{*}_{x→y} = W^B_{y} \times (\alpha^* \times d^*_{id} + \beta^* \times r^*_{id}),
\]

where \(d^*_{id}\) is the direct trust evaluation value of \(A\) about \(B\), \(r^*_{id}\) is the comprehensive combined recommended trust evaluation value of each recommender’s individual recommended trust with \(B\), \(\alpha^*\) and \(\beta^*\) are the weights of \(d^*_{id}\) and \(r^*_{id}\), respectively, and \(W^B_{y}\) is the trust weight of entity \(y\) occupying resource domain \(B\).

Detailed algorithms for evaluating \(d_{id}\) and \(r_{id}\) can be found in our previous research (Tao et al. 2007).

Set \(T^c_{x→y}\) to be the lowest trust degree of \(x\)’s requirement to \(y\), during the resource service transaction. If

\[
T_{x→y} < T^c_{x→y},
\]

then the system determines that a Trust_Failure has occurred.

- **Detection of an App_DesignCode_Failure and an App_AccessRight_Failure.** We employ the Globus GRAM (Grid Resource Allocation and Management) service, i.e. GRAM callbacks, to detect an App_DesignCode_Failure and an App_AccessRight_Failure. GRAM callbacks notify submission failures that include connection, authentication and authorisation, RSL (Resource Specification Language) parsing, and executable or input staging (Huedo et al. 2006).
6. MGrid failure recovery based on ECA rules

In this paper, we introduce a novel idea in which ECA (Event–Condition–Action) are used to provide a failure recovery service. ECA rules are one way of implementing this kind of functionality (Papamarkos et al. 2006). The reason we use ECA rules is due to the following advantages (Khanli and Analoui 2008).

- ECA rules allow this functionality to be specified and managed within a rule base rather than being dispersed in diverse programs, thus enhancing the modularity, maintainability, and extensibility of applications.
- ECA rules have a high-level, declarative syntax, and are thus amenable to analysis and optimisation techniques which cannot be applied if the same functionality is expressed directly in application code.
- ECA rules are a generic mechanism that can abstract a wide variety of reactive behaviours, in contrast to an application code that is typically specialised to a particular kind of reactive scenario.

Referring to the structure of traditional ECA rules, the proposed architecture for processing a failure during resource service scheduling based on ECA rules is shown in Figure 3. Its working flow is described as follows. During the MGrid resource service scheduling process, the failure detector will detect that a failure has occurred according to the mechanisms and algorithms described in Section 5. The event detector embedded in the scheduling system waits for an event (i.e. a detected failure), then evaluates and checks if the condition of the triggered rule is satisfied. When the condition is satisfied and the rule has not been marked for dynamic conflict, the corresponding action is executed immediately.

An ECA rule has the general syntax of

**On event If Condition Do Action.**

- An event is an incident that triggers a rule. In this paper, it is classified into four primitive events, namely the VL event (VL_Event), the resource service event (RS_Event), the task event (Task_Event), and the application event (App_Event).

![Figure 3. ECA-based MGrid resource service scheduling failure recovery.](image-url)
(RS_Event), the task event (Task_Event) and the application event (App_Event). The VL_Event is generated from VL-related failures, such as a VL_Disconnect_Failure and a Bandwidth_Failure. The RS_Event is generated from resource-service-related failures, such as an RS_Quit_Failure, an RS_Overload_Failure, and an RS_Composition_Failure. The Task_Event is generated from task-related failures, such as a Task_Cancel_Failure, a Task_Suspension_Failure, and a Task_RS_Mismatch_Failure. The App_Event is generated from application-related failures, such as a Trust_Failure, an App_DesignCode_Failure, and an App_AccessRight_Failure.

- The condition of a rule is a Boolean statement that must be satisfied in order to activate a rule. It is described in terms of an XPath expression (Jung et al. 2007), and the expression in the condition may refer to values from the event definition.
- The action part contains the instruction that is executed when a triggered rule is activated.

In our proposed work, ECA rules are used directly to provide a failure recovery service during MGrid resource service scheduling.

Before introducing the corresponding failure recovery ECA rules, we first investigate the life cycle of the two main factors that cause failure in the MGrid scheduling process: resource service and task.

During the whole process of MGrid resource service scheduling, the state of a resource service can be classified into three categories: (a) ARRIVAL, a resource service is published and inserted into the RS_Advertise queue; (b) IDLE, a resource service is ready to execute a task at any time and it is inserted into the RS_Idle queue; and (c) BUSY, a resource service is executing a task and it is inserted into the RS_Busy queue. If a resource service has completed its allocated tasks, or the resource service is inappropriate for the allocated task, the resource service will then be deleted from the RS_Busy queue and inserted into the RS_Idle queue. The whole life cycle of a resource service and its state transformation in the MGrid resource service scheduling process are shown in Figure 4.

In the same way, the state of a task can be classified into four categories: (a) ARRIVAL, a task is submitted by the user and inserted into the Task_Arrival queue; (b) WAIT, a task is ready to be executed at any time when an appropriate resource service is free, and any task in the WAIT state is inserted into the Task_Wait queue; (c) RUNNING, a task is being executed on a resource service, and any task in a WAIT state is inserted into the Task_Running queue; and (d) FINISHED, a task has been completed and it is inserted into the Task_Finished queue. The whole life cycle of a task and its state transformation during MGrid resource service scheduling are shown in Figure 5.

Based on the above work, we designed the following ECA rules for the recovery of the above-mentioned failures in the MGrid resource service scheduling process.

ON Event RS_Quit_Failure (RS, task)
Do Action (delete RS from RS_Busy)
If Condition (Select RS from RS_Idle where RS.QoS satisfies task.QoS)
Do Action Match RS and task
If Condition (there is no RS in RS_Idle satisfying task)
Do Action (Delete task from Task_Running)
Then Do Action (Insert task into Task_Wait)
ON Event \texttt{RS\_Overload\_Failure (task)}
Do Action \{Insert task into Task\_Wait\}
If Condition (Select RS from RS\_Idle where RS.QoS satisfies task.QoS)
Do Action Match RS and task
If Condition (there is no RS in RS\_Idle satisfying task) $k$
Do Action \{Insert task into Task\_Wait\}

ON Event \texttt{RS\_AbilityChange\_Failure (RS, task)}
Do Action (compare RS.QoS and task.QoS)
If Condition ($RS.QoS$ satisfies task.QoS)
Do Action (Exit)
Else
{
  Do Action (Delete RS from RS_Busy)
  Then Do Action (Insert RS into RS_Idle)
  And Do Action (Select RS from RS_Idle where RS.QoS satisfies task.QoS)
  If Condition (there is no RS satisfies task)
  {Do Action (Delete task from Task_Running)
   Then Do Action (Insert task into Task_Wait)}
}

ON Event Task_Cancel_Failure (RS, task)
  Do Action (Delete task from Task_Running)
  Do Action (Delete RS from RS_Busy)
  Do Action (Insert RS into RS_Idle)

ON Event Task_Suspension_Failure (task)
  If Condition (task.SuspensionTime > task.SuspenensionThreshold)
  Do Action (Select RS form RS_Idle where RS.QoS satisfies task.QoS)
  If Condition (there is no RS satisfies task)
  Do Action (Delete task from Task_Running)
  Then Do Action (Insert task into Task_Wait)

ON Event Task_RS_Mismatch_Failure (RS, task)
  Do Action (Delete RS from RS_Busy)
  Then Do Action (Insert RS into RS_Idle)
  Do Action (Select RS form RS_Idle where RS.QoS satisfies task.QoS)
  If Condition (there is no RS satisfying task)
  Do Action (Delete task from Task_Running)
  Then Do Action (Insert task into Task_Wait)

ON Event Task_RequireChange_Failure (RS, task)
  Do Action (compare RS.QoS and task.QoS)
  If Condition (RS.QoS satisfies task.QoS)
  Do Action (Exit)
  Else Do Action (Delete RS from RS_Busy)
  Then Do Action (Insert RS into RS_Idle)
  And Do Action (Select RS from RS_Idle where RS.QoS satisfies task.QoS)
  If Condition (there is no RS satisfies task)
  Do Action (Delete task from Task_Running)
  Then Do Action (Insert task into Task_Wait)

ON Event Trust_Failure (x, y, task, RS) //here x denote the resource service demander (RSD) or the entity who submits the task, and y denotes the resource service provider (RSP) who provides RS
  If condition (\( T_{x \rightarrow y} < T_{y \rightarrow x}^c \))
  Do Action (Delete RS from RS_Busy)
  Then Do Action (Insert RS into RS_Idle)
  Do Action (Select a new RS belonging to RSP z from RS_Idle where \( T_{x \rightarrow z} \geq T_{x \rightarrow z}^c \))
  If condition (there is no RS satisfying the task of x)
  Do Action (Delete task from Task_Running)
Then Do Action (Insert task into Task_Wait)

The above ECA rules for failure recovery are examples, but not an exhaustive list.

7. Implementation and simulation

In order to verify the rationality and usability of our proposed method, we combine our failure detection and recovery method with the Min-min (Tracy et al. 2001) scheduling method, and call it the F-Min-min scheduling method. The Min-min scheduling method is described as follows.

- The Min-min heuristic begins with a set of tasks. Then the set of minimum completion times for each task is found. Next, the task with the overall minimum completion time from the task set is selected and assigned to the corresponding resource services. Lastly, the newly assigned task is removed from the task set, and the process is repeated until all tasks are assigned. Min-min is based on the minimum completion time (Tracy et al. 2001).

We compare our method (i.e. F-Min-min) with Min-min with respect to the number of failures of the task, the success rate, and the total execution time. We first take 100–500 magnetic bearing parameterised design requests as the tasks. The requested parameters are generated randomly by computer. We select 100 resource services available in the experimental system ‘Magnetic Bearing Resource & Service Sharing Platform under MGrid’ (MBRSSP-MGrid) as the resource service providers (RSPs). MBRSSP-MGrid was developed in our laboratory for the sharing and collaborative working of the various resources needed in the development process of a magnetic bearing system. In MBRSSP-MGrid, any user can publish their idle resources, including equipment resources, software resources, human resources, application resources, technique resources, and service resources, through the resource publication centre of MBRSSP-MGrid (Tao et al. 2008). The user can also search for resources or services they require, e.g. remote parameterised design services for magnetic bearings, as shown in Figure 6, through the resources and service optimal allocation centre in MBRSSP-MGrid (Tao et al. 2008).

7.1 Comparison of success rate

In these experiments the four kinds of failures proposed in Section 3 were artificially generated. For example, during the execution of tasks, some resource service providers or demanders were disconnected from the network by cutting off the network wire; several running tasks were suspended by using the stop command on task manager; and the resource service was overloaded with an artificially generated load. The distributions of each kind of failure in these experiments are shown in Figure 7.

Comparisons of the total number of failures and number of detected failures for the F-Min-min method are shown in Figure 8 for resource service scheduling. In Figure 7, we found that more than 80% of the total failures can be detected using the proposed failure detection method. This demonstrates that the proposed failure detection methods are effective.

A comparison of the number of task failures and success rate between F-Min-min and Min-min is shown in Figures 9 and 10, respectively. From these figures we can see that the F-Min-min scheduling method has a smaller task failure number than the Min-min
scheduling method, and has a higher success rate than Min-min. The reason for this is that the F-Min-min scheduling method takes into account failures that are generated during the scheduling process, and has a failure detection and recovery service. The F-Min-min method can detect 80% of the failures occurring during the scheduling process, and most
of the detected failures can be recovered using the methods proposed in Section 5. Hence, \textit{F-Min-min} has a higher success rate and smaller failure task number than the simple \textit{Min-min} method. This demonstrates that our proposed failure detection and recovery method is effective and useful.
Comparison of execution time

In order to further test the effectiveness of our proposed method under different tasks and resource services, in addition to the above simple task with a single resource service request, we considered more complicated tasks in the development process of a magnetic bearing system as experimental objects in MBRSSP-MGrid. We selected 10 tasks, of which the decomposed subtasks vary from 1 to 10 with an increment of 1 as experimental objects such as on-line analysis and simulation of the axial magnetic bearing system ($T_{\text{magnetic}}$), which can be decomposed into five subtasks (Tao 2008): (a) the axial magnetic bearing structure parameterised design service; (b) the mechanics analysis and simulation service; (c) the temperature field simulation and analysis service; (d) the magnetic field simulation and analysis service; and (e) the controlling system simulation and analysis service. The system has to invoke five different resource services in a certain sequence to execute $T_{\text{magnetic}}$.

During the scheduling process of these tasks, the failures proposed in Section 3 are artificially generated in the same way as mentioned above. We selected the average execution time of these tasks as the index to evaluate the effectiveness of our proposed method.

Figure 11 shows a comparison of the execution time between the use of the Min-min algorithms and performing failure recovery through task scheduling when failures occur (namely F-Min-min). Each result is an average of 20 successful test cases. In this paper, the total execution time of a task depends on the longest execution time of its subtasks. It consists of the resource service selecting time, composing time, QoS information processing time, task executing time, etc., during the scheduling process. In Figure 11, we can see that the total execution time decreases significantly due to failure recovery using the F-Min-min method. The reason for this is that the simple Min-min method has no additional failure detection and recovery service or function, and subtasks (or tasks) are stopped or suspended and have to wait to be restarted until the system assigns a new resource service to it, hence the total execution time becomes longer due to the unforeseen waiting time. However, there are additional failure detection and recovery services in F-Min-min, and when a failure occurs on a subtask (or a task), the system can detect and recovery it as soon as possible, and the waiting time is correspondingly much shorter than that in Min-min. Therefore, the total execution time of F-Min-min is clearly shorter than Min-min. This demonstrates that the proposed failure detection and recovery methods are effective and useful, and they are useful in shortening the execution time of a complicated task and improving the quality of service of scheduling in MGrid.
8. Conclusion
The topic of MGrid is new and current work on MGrid primarily concentrates on concepts, architecture, the application prototype platform, application foreground, etc. Failure detection and recovery, which are key for realising dynamic MGrid resource service scheduling, are mostly absent and have not been effectively addressed. In this paper, the potential failures that would be generated during MGrid resource service scheduling are investigated. Thirteen failures are defined in detail, and are classified into four classifications: (a) virtual-link-related failures; (b) resource-service-related failures; (c) task-related failures; and (d) application-related failures. A failure management system is proposed that provides a failure-tolerance service in MGrid resource service scheduling. Corresponding detection mechanisms and methods for each defined failure are presented in detail, associated with the corresponding failure recovery methods. The implementation and simulation results indicate that the proposed approaches are sound for promoting the resource service scheduling success rate.

In the future we plan to carry out more experimental work, and implement our resource service failure detection and recovery methods as middleware and perform various experiments to measure its efficiency in MGrid resource service scheduling. In addition, we will investigate a way by which failure detection and recovery can be used for resource service optimal selection and allocation, and task migration in heterogeneous MGrid environments.

Acknowledgements
The first author wishes to acknowledge the financial support of the Excellent Doctoral Dissertation Fund of Wuhan University of Technology (WHUT). This paper is supported by the Hubei Digital Manufacturing Key Laboratory Opening Fund Project: Research on resource service search and optimal-selection theories and experiments in manufacturing grid systems (No. SZ0621) and the National Nature Science Fund Project of China: Research on new theories and new technologies for network-based digital manufacturing environments (No. 50620130441). We would also like to express our great appreciation to the valuable comments made by four anonymous reviewers and editors of this journal.

References


