



## **WestminsterResearch**

<http://www.wmin.ac.uk/westminsterresearch>

### **Fault-tolerant behavior in state-of-the-art grid workflow management systems.**

**Kassian Plankensteiner<sup>1</sup>**  
**Radu Prodan<sup>1</sup>**  
**Thomas Fahringer<sup>1</sup>**  
**Attila Kertesz<sup>2</sup>**  
**Peter Kacsuk<sup>2\*</sup>**

<sup>1</sup> Institute for Computer Science, University of Innsbruck

<sup>2</sup> MTA SZTAKI Computer and Automation Research Institute, H-1518  
Budapest, P.O. Box 63, Hungary

\* Peter Kacsuk is a Professor in the Centre for Parallel Computing, School of Informatics, University of Westminster

This is a reproduction of CoreGRID Technical Report Number TR-0091, October 18, 2007 and is reprinted here with permission.

The report is available on the CoreGRID website, at:

<http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0091.pdf>

---

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners. Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

---

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch.  
(<http://www.wmin.ac.uk/westminsterresearch>).

In case of abuse or copyright appearing without permission e-mail [wattsn@wmin.ac.uk](mailto:wattsn@wmin.ac.uk).

## **Fault-tolerant behavior in state-of-the-art Grid Workflow Management Systems**

*Kassian Plankensteiner, Radu Prodan, Thomas Fahringer*  
{kassian.plankensteiner, radu, tf}@dps.uibk.ac.at  
*Institute for Computer Science*  
*University of Innsbruck*  
*Technikerstr. 21a*  
*A-6020 Innsbruck, Austria*

*Attila Kertész, Péter Kacsuk*  
{attila.kertesz, kacsuk}@sztaki.hu  
*MTA SZTAKI Computer and Automation Research Institute*  
*H-1518 Budapest, P.O. Box 63, Hungary*



CoreGRID Technical Report  
Number TR-0091  
October 18, 2007

Institute on Grid Information, Resource and  
Workflow Monitoring Services

CoreGRID - Network of Excellence  
URL: <http://www.coregrid.net>

# Fault-tolerant behavior in state-of-the-art Grid Workflow Management Systems

Kassian Plankensteiner, Radu Prodan, Thomas Fahringer  
{kassian.plankensteiner, radu, tf}@dps.uibk.ac.at  
Institute for Computer Science  
University of Innsbruck  
Technikerstr. 21a  
A-6020 Innsbruck, Austria

Attila Kertész, Péter Kacsuk  
{attila.kertesz, kacsuk}@sztaki.hu  
MTA SZTAKI Computer and Automation Research Institute  
H-1518 Budapest, P.O. Box 63, Hungary

*CoreGRID TR-0091*

October 18, 2007

## Abstract

While the workflow paradigm, emerged from the field of business processes, has been proven to be the most successful paradigm for creating scientific applications for execution also on Grid infrastructures, most of the current Grid workflow management systems still cannot deliver the quality, robustness and reliability that are needed for widespread acceptance as tools used on a day-to-day basis for scientists from a multitude of scientific fields. This paper introduces the current state of the art in fault tolerance techniques for Grid workflow systems. The examined categories and the summary of current solutions reveal future directions in this area and help to guide research towards open issues.

## 1 Introduction

In the past years, the workflow paradigm has emerged as the most successful paradigm for creating scientific applications for execution on Grid infrastructures. Numerous groups from all over the world have created a plethora of Grid workflow description languages and Grid workflow management systems, all of them sharing the same basic goals: to create a system that can be used to easily and reliably execute workflow applications on huge heterogeneous Grid systems.

Up to now, most of the existing Grid workflow systems still cannot deliver the quality, robustness and reliability that is needed for widespread acceptance as tools used on a day-to-day basis for scientists from a multitude of scientific fields. The scientists typically want to use the grid to compute solutions for complex problems, potentially utilizing thousands of resources for workflows that can run for several hours, days or even weeks. With a system that has a low tolerance for faults, the users will regularly be confronted with a situation that makes them lose days or even weeks of valuable computation time because the system could not recover from a fault that happened before the successful completion of their workflow applications. This is, of course, intolerable for anyone trying to effectively use the

---

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

Grid, and makes scientists accept a slower solution that only uses their own computing resources because of a higher reliability and controllability of these systems.

Due to the heterogeneous and distributed nature of Grid systems, faults inevitably happen. This problem can only be overcome by highly fault-tolerant systems. The reasons for faults in a Grid environment are manifold: the geographically widespread nature encompassing multiple autonomous administrative domains, variations in the configuration of the different systems, overstrained resources that may stop responding or show unpredictable behavior, faults in the network infrastructure that connects the systems, hardware failures and systems running out of memory or disk space are just some of the possible sources of faults.

This report shows the current state of the art in fault tolerance techniques for Grid workflow systems. We will show which faults can be detected, prevented and recovered by which current Grid workflow system, the areas where the current systems are planned to improve and which areas are still in need of more research effort.

## 2 Questionnaires for surveying

### 2.1 Related work

The taxonomy by Buyya et al. in [3] introduces a general view of existing workflow managing solutions. A part of it focuses on fault tolerance, where they use a task- and workflow-level division. We used and further extended these categories, keeping the scope on fault tolerance. While the presented taxonomy reveals a glimpse of the overviewed systems, in this paper we give a detailed description and comparison of their properties.

In [1], Hwang et al. propose a multi-layered approach for fault tolerance in workflows. They segment the techniques into task-level and workflow-level techniques. The former tries to hide faults that happen during the execution of single tasks at the workflow-level, while the latter manipulates the structure of the workflow to deal with faults dynamically.

### 2.2 Questionnaire conformation and description

To build a general and objective vision of state-of-the-art fault tolerance support in grid workflow management systems, we not only have made an extensive literature review, but also have sent out a detailed questionnaire (see Appendix).

On the first page we asked for general information: contact data and history on fault tolerance. We divided the target questions into two main categories: fault detection on one hand and fault recovery and prevention on the other. In both categories we identified several layers where detection as well as recovery and prevention can exist. Faults can be detected at Hardware, Operating System, Middleware, Task, Workflow or User level. At the lowest level, the Hardware level, machine crashes and network connectivity errors can happen. At the level of Operating Systems, tasks may run out of memory or disk space, or exceed CPU time limits or disk quota. Other faults like network congestion or file non-existence can also happen. One level higher at the Middleware, we could find non-responding services, probably caused by too many concurrent requests. Authentication, file staging or job submission failures can happen, and submitted jobs could hang in local queues, or even be lost before reaching the local resource manager. At the level of Tasks, job-related faults can happen, like deadlock, livelock, memory leak, uncaught exceptions, missing shared libraries or job crashes, even incorrect output results could be produced. At Workflow level, failures can occur in data movement or infinite loops in dynamic workflows. Incorrect or not available input data could also produce faults. Finally, at the highest level, the User level, user-definable exceptions and assertions can cause errors (for example the users can define conditions, such as the output file size should not be bigger than 5 MB). Beside all these attributes, the developers had the opportunity to add new ones, like incorrect job description format at the Middleware level. We created two tables questioning the above mentioned attributes. The first one answers whether the system can detect and cope with these faults (prevent or recover). The second one is used to name the service or component the system uses to detect the listed faults.

In the fault prevention and recovery tables, we distinguished among three abstraction levels. The treatment mechanisms can act at Task-, Workflow- or User-level. At the Task level, recovery is used when a failed job is restarted on the same remote resource or resubmitted to another one. Generally it is simple to implement this technique; upon detecting a failure, the task is rescheduled to either the same or to another resource for another try. Resubmission can cause significant overheads if the following tasks have to wait for the completion of the failed task. Saving checkpoints and restarting later or even migrating jobs can be a good prevention and recovery mechanism. This technique stores all

the intermediate data of a task that is needed to restore the task to the current state. This allows for migration of a task to another system in case of failure: it can resume execution from the last checkpoint, unlike simple resubmissions, where jobs should be started over from the beginning.

Task replication can prevent resource failures, while alternate task creation can recover from internal task failures (in this case another task implementation is executed). On failures of the task manager itself, recovery means restarting the service or choosing another one. Finally resource reliability measurements can also prevent job execution faults. At Workflow level, redundancy, data and workflow replication can prevent faults. Redundancy, sometimes called replication in related work, executes one task concurrently on several resources, assuming that one of the tasks will finish without a failure. It can cause overhead by occupying more resources than necessary, but guarantees failure-free execution as long as at least one task does not fail. Light- and Heavy-weight checkpointing can also be used for both prevention and recovery. Generally this technique can be used to save an intermediate state of a whole workflow for a restart at a later point in time. Light-weight checkpointing saves only the current location of the intermediate data, not the data itself. It is fast, but restarting can only work as long as the intermediate data is available at its original location. Heavy-weight checkpointing saves all the intermediate data to a place, where it can be kept as long as it is needed. In case of a failure of a service that is needed to manage the workflow execution, a system can use the Management Service Redundancy technique that chooses another equivalent service or restart the faulty service to resume operation. The transaction and rollback mechanisms can be used for the same reason. Should the workflow manager itself crash, restarting the service or choosing another manager means a high-level recovery option. Finally the task manager reliability measurements can prevent choosing managers that are known to be unreliable.

At the highest level, the User level, user-defined exceptions can be taken into account to validate proper execution. The questionnaire also contained two tables for this section: the first is used to tell whether the listed mechanism is supported or not, the second is for naming the service that handles the faults.

## **3 Evaluation**

### **3.1 ASKALON**

The goal of ASKALON [2] is to simplify the development and optimization of applications that can harness the power of Grid computing. This project crafts a novel environment based on new innovative tools, services, and methodologies to make Grid application development and optimization for real applications an everyday practice. The system is centered around a set of high-level services for transparent and effective Grid access, including a Scheduler for optimized mapping of workflows onto the Grid, an Enactment Engine for application execution, a Resource Manager covering both computers and application components, and a Performance Prediction service based on training phase and statistical methods. ASKALON builds upon its own XML-based workflow language called AGWL that not only covers the usual DAG-based workflow approach, but adds complex constructs such as parallel loops and conditional statements such as switch and if/then/else. At the hardware level, the system detects and recovers from machine crashes and network failures. Askalon detects faults like exceeded disk quota, out of disk space and file not found faults at the OS level and can recover from the first two of them. On the middleware-level, the system is able to detect failed authentication, failed job submissions, unreachable services and file staging failures, and is able to recover from a failed job submission. The Task level faults are not detected by the system, and therefore it cannot recover from them either. On the workflow-level, unavailable input data and data movement faults can be detected, and the system can also recover from data movement faults. Askalon does not support user-definable exceptions or assertions in its current version, but this is planned for a future release.

### **3.2 Chemomentum**

The Chemomentum project [8] takes up and enhances state-of-the-art Grid technologies and applies them to real-world challenges in computational chemistry and related application areas. It helps the transformation of computing paradigms in these areas towards collaborative research and Grid computing. The Chemomentum system is currently under development and will build upon UNICORE 6 [9], the web-services version of UNICORE released in 2007, adding a two-layer workflow engine on top of the UNICORE 6 middleware. The top layer, called process engine, deals with the high-level workflow concepts, while the lower layer, called service orchestrator, deals directly with lower-level concepts such as running jobs and moving files. The actual act of processing of the tasks themselves is given to the underlying UNICORE 6 system, using XNJS as the entity that accesses the different Grid sites batch-

and file-systems. Fault tolerance is a very important topic for Chemomentum. At the low level, machine and network failures can be detected by the UNICORE 6 infrastructure; recovery from these faults is planned for a future version of Chemomentum. Chemomentum cannot detect faults at the OS level, but the system is able to detect authentication failures, failed job submissions, non-responding or non-reachable services and file-staging failures at the middleware level. It can recover from failed job submissions (handled by the service orchestrator) and in case a service is unreachable. Taking a look at the task level, the system is able to detect crashed jobs, but cannot recover from this fault. At Workflow level, the system can detect unavailable input data and data movement failures (handled by the service orchestrator), but cannot prevent them or recover from such a fault. User definable exceptions or assertions are not supported in Chemomentum. It relies heavily on the underlying UNICORE 6 middleware for fault detection and recovery; the Chemomentum system itself mainly adds higher-level fault detection and recovery features on top.

### 3.3 Escogitare WFMS

One of the main targets of the Escogitare project [10] is to enable agriculture scientists, located in several CRA institutes spread all over Italy, to conduct bioinformatics and geoinformatics experiments using a workflow management system that is able to select data and instruments installed in several laboratories in a transparent and secure way. The project has been started 2 years ago. The enactor uses the BPEL language [11] for describing workflows. The system mainly relies on the “catch” operation of BPEL, and fault tolerance is not the most important feature. At the lowest, Hardware level, machine and network failures can be detected and prevented by MDS4 [6], and they are planning to support recovery of these faults in the future version of the enactor. Operating system faults can be detected by reports of the invoked Web Service; they also work on recovery for future releases. At Middleware level, non-responding or non-reachable services are detected, the recovery is being investigated for future version, as well as authentication failure detection. Task execution faults are recognized by the responsible Web Service reports, but recovery is still under development. At Workflow level, loops, input errors and non-availability are detected. At the highest level user definable exceptions are handled by the BPEL “catch” and “catchAll” construct, therefore they are detected, and recovery is planned to be supported in a future version of the system. All the BPEL language related faults are handled by the ActiveBPEL 2.0 Engine [11] used by Escogitare. Though they rely on the detection and prevention support of their utilized components in the current version of the system, recovery is not available. Since the ActiveBPEL Engine gives the framework for recovery features, for the future version a higher level fault tolerance is under development. At the Workflow level redundancy, data replication and the transaction/rollback mechanisms are planned to be supported, while at the Task level resubmissions are investigated.

### 3.4 GWEE

The Grid Workflow Execution Engine (GWEE) [12] is developed at Umea University in Sweden. The workflow engine is implemented as a WSRF service and its fault detection and recovery is dependent on the grid service container provided by Globus Toolkit version 4. The workflow engine itself is independent of client applications as well as middleware. The developers believe that this kind of support must be provided by the client application or portal that is using the GWEE for state management and dependency control. When a fault occurs, the workflow engine receives the signal and propagates the fault to the client application. The client application is responsible for taking an appropriate action in GWEE. That is also the reason why the engine itself is not able to detect, prevent or recover from faults at the Hardware- or Operating System-level. At the middleware-level, the GWEE can detect and recover from failed authentications, failed job submissions and file staging failures using the Enactor/Executor plugin and the Workflow service. At Task level, the system can detect and recover from a crashed job, but not from other faults like memory leaks, uncaught exceptions, deadlocks or missing shared libraries. On the workflow level, failed data movement can be detected and recovered from, but infinite loops, unavailable input data and input errors remain undetected. User-definable exceptions and assertions are not supported directly by GWEE; the system leaves that to the client applications. At the workflow-level, GWEE implements fault recovery techniques like workflow level checkpointing (light- and heavy-weight) as well as transaction/rollback mechanisms, it can pause and resume the workflow using GT4-provided functionality. Data and workflow replication is not handled by the engine itself, but has to be provided by high-level services or client applications on top of GWEE. At Task and User-level, GWEE does not employ fault recovery or prevention techniques.

### 3.5 GWES

The Grid Workflow Execution Service [13] is the workflow enactment engine of the K-Wf Grid [14], which coordinates the creation and execution process of Grid workflows. It was first announced in 2003. The new version that supports high-level fault tolerance was released this year, therefore fault tolerance is an important feature of the system. The GWES uses High Level Petri Nets for modeling workflows, which are very suitable regarding fault management issues, as they contain the whole workflow state. At the lowest level, machine crashes and network failures can be detected and the system is able to recover with the help of the ResourceUpdater component of GWES. At the Operating system level, if a file is not found, it can be detected by RFT [5], and the system can recover using its activity-handling component. At Middleware level, authentication, file staging and job submission failures can be detected through WS-GRAM [6], and the system is able to recover from these failures. When a service is not reachable or not available, it is detected, and the recovery is done by the ResourceUpdater. At Task level the correctness of output data is validated, job crashes can be detected by WS-GRAM or through Web Service responses, and the system is able to recover. At the top Workflow and User levels all the listed faults can be detected and handled with recovery. In general, the GWES handles faults mainly at the Workflow and Task level. This means, if an activity fails on one host, then it is repeated on another host, depending on the fault type. If no host is left as candidate, then a new resource matching is done to find new resources. In addition, GWES features some basic monitoring of hardware, software and services. The results of this monitoring are stored in a XML database which is used as basis for the resource matching. Due to this technique, the system can recover from many fault types, even without knowing the specific reason of the fault. In addition to the "traditional batch Grid" fault types, the GWES takes care about the fault tolerance in a pure Web Service environment, evaluating and responding to SOAP [15] faults returned by a remote Web Service method call. Focusing on recovery and prevention, at Workflow level, redundancy, workflow replication and checkpointing are handled by the system itself. Heavy-weight workflow checkpointing is only supported for workflows enacting pure Web Services (SOAP). Data replication mostly depends on the underlying data layer or file system, and is not handled by GWES itself (SRB, gpfs or user-defined). At Task level, job retry and resubmission is supported by the system, and resource reliability measurement is done utilizing a simple score mechanism (ala eBay [16]). Finally, user defined exception handling is also supported. The system supports user-defined fault management in a way that the user can insert additional sub-workflows in order to evaluate and react on specific workflow activity results (exit status, data, side effects, ...). This makes it possible to react on application specific faults (e.g. restart licence server if application returns some exit code).

### 3.6 Pegasus

Pegasus [17] (Planning for Execution in Grids) is a workflow mapping engine first released in 2001. It bridges the scientific domain and the execution environment by automatically mapping the high-level workflow descriptions onto distributed infrastructures. At the lowest, Hardware and Operating System levels, it can detect exceeding CPU time limit and file non-existence, and by the help of DAGMan [18] it can recover from machine crashes and network failures. They find fault tolerance important, and as a future step they plan to support recovery from running out of disk space or exceeding disk quota. At the level of Middleware, it detects authentication, file staging and job submission faults, and the system is able to recover with DAGMan. At Task and Workflow levels job crashes and input unavailability are detected, data movement faults can also be treated with recovery, again with DAGMan. Regarding recovery and prevention, at Workflow level redundancy is used and light-weight checkpoints are supported by Pegasus itself. At Task level retries, resubmissions and checkpointing are supported, task migration, replication and alternate task creation are planned for future version.

### 3.7 P-GRADE WFMS

The P-GRADE Grid Portal [4] is a web based, service rich environment for the development, execution and monitoring of workflows and workflow based parameter studies on various grid platforms, which is available since 4 years. Its Workflow Manager is based on DAGMan, and during the development of the new version they take into account advanced fault-tolerant features. At the lowest level, the Hardware level, machine crashes and network failures are detected, and the system is able to recover from these faults. At the Operating System and Middleware levels, running out of memory, disk space or exceeding disk quota can be recognized by the manager, and it is also able to recover. Missing files can be prevented, detected and also treated with recovery. Service unavailability, authentication, file staging, job submission, job hanging in the queue of a job manager or being lost before reaching it can be detected

by the enactor with the help of Resource Managers (GRAM [5], EGEE WMS [7]), and recovery from these faults are also supported. Incorrect job descriptions can also be prevented by the Workflow Manager. At Task level, job crashes are detected, and recovery is also supported for faults caused by missing shared libraries. At Workflow level, input unavailability can be prevented, in case of input loss it can be detected and treated with recovery, which is also the case for other data movement failures. Focusing on prevention and recovery, at Workflow level, redundancy can only be done manually, but light-weight checkpointing and restarting of the workflow manager on failure is fully supported. At Task level, checkpointing at OS-level is supported by PGRADE. Retries and resubmissions are supported by task managers (EGEE WMS), and task management service selection is supported by the enactor. Restarting of task managers and resource reliability measurement are planned to be supported in the future version.

### **3.8 ProActive**

ProActive [19] is a middleware for parallel, distributed and multi-threaded computing. The ProActive Skeleton Framework [19] is available since 2 years. Fault tolerance is an important issue for the system developers. Many of the fault tolerance issues are handled by the lower level middleware, in this case ProActive. The Calcium framework is mainly addressed as a Skeleton Framework, but generally fits the definitions stated in the questionnaire. The corresponding components of the architecture are the following: The ProActive Executor is the unit of logic that executes the program on the computation resources. The ProActive Enactor schedules tasks according to the application's semantics. The ProActive Core provides relations with other services. At the level of Hardware, the ProActive Core is responsible for detecting and recovering machine and network failures. The developers plan to address prevention issues for these faults in a future version of the system. At the Operating System level, the ProActive Executor detects missing files, running out of memory or disk space and disk quota exceedance. The prevention and recovery of these failures are investigated for future releases, as well as the detection and recovery of network congestion and CPU time limit violation. At the Middleware level service unavailability, authentication, file staging and job submission faults can be recognized by the Proactive Core, but recovery or prevention will only be available in a future version. The rest of the listed faults (job loss, pending and concurrent request overload) are planned to be supported later. At Task level the ProActive Enactor detects uncaught exceptions and incorrect output data. The rest of the faults listed at this level are not supported yet, but they are planned to be treated at all stages in a future version. At Workflow level data movement, input errors and unavailability are detected, but prevention and recovery issues are under development, as well as infinite loop elimination. Finally at the highest level user-defined exceptions are detected by the Executor, prevention or recovery is planned for future releases together with user assertions. Regarding prevention and recovery, at Workflow level the transaction/rollback mechanisms are supported by ProActive Core, redundancy usage and workflow management service restart are scheduled for next releases. At Task level, retries, resubmissions and checkpointing are supported by the Enactor and the Core. Task migration and task manager service restart are under investigation.

### **3.9 Triana**

The Triana problem solving environment [20] is an open source problem solving environment developed at Cardiff University that combines an intuitive visual interface with powerful data analysis tools. Already used by scientists for a range of tasks, such as signal, text and image processing, Triana includes a large library of pre-written analysis tools and the ability for users to easily integrate their own tools. It exists since 5 years. They regard fault tolerance as an important issue. Support for fault tolerance is generally user driven and interactive in Triana with little automated systems. For example, faults will generally cause workflow execution to halt, display a warning or dialog, and allow the user to modify the workflow before continuing execution. At the lowest level, machine caches and network errors are recognized by GridLab GAT [21] and the Triana Engine [20] respectively, but recovering from these faults or preventing them is only planned for future versions. Looking at the Operating System level, missing files are detected by the Engine and GAT, prevention or recovery is under investigation for future releases, as well as the other listed faults at this level. At the Middleware and Task levels, all the listed faults can be detected by the Engine or GAT, except for deadlock, livelock and memory leaks. All the listed faults at both levels will be treated in the future version with prevention and recovery. At the next level, the Workflow level, data movement and input availability errors are detected by the Triana Engine. The listed faults together with User level faults are planned to be handled in future releases. Focusing on prevention and recovery, at Workflow level light-weight checkpointing and the restart or selection of workflow management services are currently supported, the rest of the features are planned to be supported later. Regarding Task level, retries, resubmissions, alternate task creations, restarts or selection of task managers are already

supported by the Engine, and the rest of the listed features in this category are under investigation for future releases.

### 3.10 Unicore 5

UNICORE (Uniform Interface to Computing Resources) [9] offers a ready-to-run Grid system including client and server software. UNICORE makes distributed computing and data resources available in a seamless and secure way in intranets and the internet. The workflow management of Unicore 5 has a history of 5 years. At the lowest levels, the Hardware and Operating System levels, all the listed faults can be detected by the NJS (Scheduler) [9]. At Middleware level, the authentication is managed by Unicore Gateways, and all the listed faults are detected by NJS. The prevention of having too many concurrent requests and the recovery from job loss and service unavailability are possible. Preventing methods for service unavailability and file staging errors are investigated, as well as recovering from staging faults. At Task level, memory leaks, uncaught exceptions, deadlocks and livelocks can be detected by TSI (Target System Interface) [9]. Missing shared libraries and job crashes are detected by NJS, but the recovery from job crashes will only be supported in a future version. At the highest levels, the Workflow- and User levels, data movement, input availability failures and user-defined exceptions are recognized by NJS. Regarding prevention, at Workflow and Task levels the Site Monitor (SIMON) [9] is responsible for task manager and resource reliability measurements, while regarding recovery, the NJS is used to retry failed jobs on the same resource.

## 4 Summary and comparison

In this section we summarize the results of the questionnaire evaluation. Figure 1 shows the percentage of the faults in each of the categories that are detected by a Grid workflow system on average.

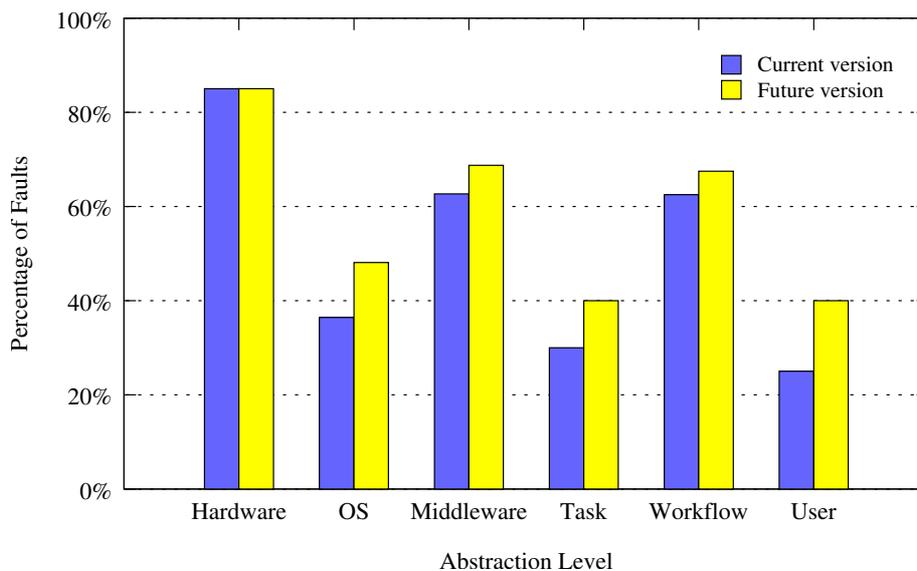


Figure 1: Average fault detection

We can see that Hardware-level faults (Machine crashed/down, Network down) can generally be successfully detected by current workflow systems. When it comes to the other categories, the situation is quite different. On the OS-level, only 37% of the faults (Disk quota exceeded, Out of memory, Out of disk space, File not found, Network congestion, CPU time limit exceeded) are currently detected on average. Detection of the faults on middleware level (Authentication failed, Job submission failed, Job hanging in the queue of the local resource manager, Job lost before reaching the local resource manager, Too many concurrent requests, Service not reachable/not responding, File staging failure) is more common, an average of 62.8% of these faults can be detected by current Grid workflow systems, which is almost the same within Workflow-level faults (Infinite loop, Input data not available, Input error, Data movement failed) with 62.5%. The worst fault detection can be seen on the Task-level (Memory leak, Uncaught exception, Dead-

lock/Livelock, Incorrect output data, Missing shared libraries, Job crashes) and User-level (User-definable exceptions, User-definable assertions), where only 30% (task-level) and 25% (user-level) of the faults are detected on average.

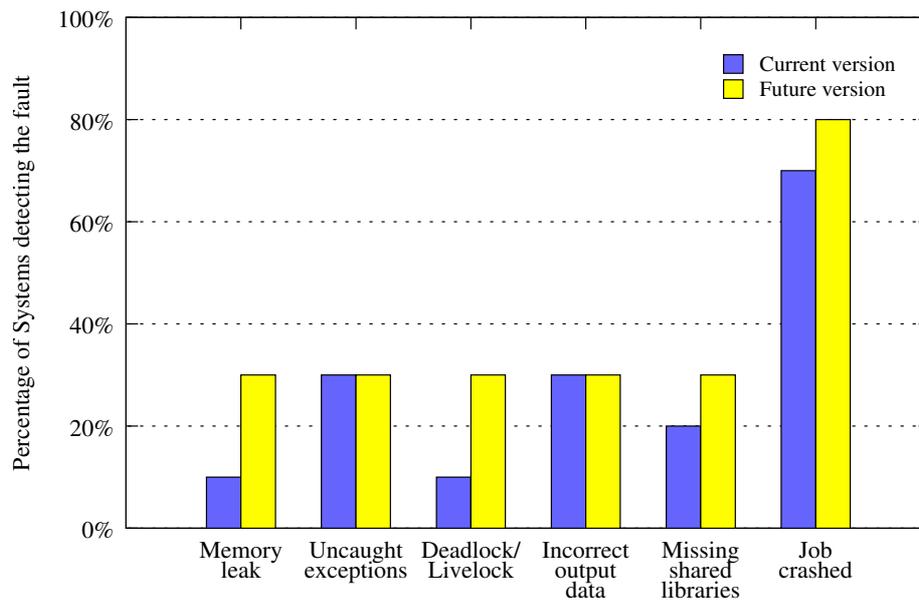


Figure 2: Percentage of systems that can detect the various task level faults

More insight into the reasons for such a bad value on the task-level faults can be seen in Figure 2. It shows for every task-level fault, the percentage of the systems that are able to detect the fault. We can see that only one out of the ten systems studied is able to detect a memory leak that happens inside of an executed task. Deadlocks/Livelocks are also only detected by one of the studied systems. While uncaught exceptions (e.g. numerical exception) and incorrect output data are detected by three systems, missing shared libraries are only detected by two of the systems. Detection of a crashed job seems to be a problem that most of the systems are able to solve, seven out of ten systems implement this functionality, with one more system where this feature is planned for a future version.

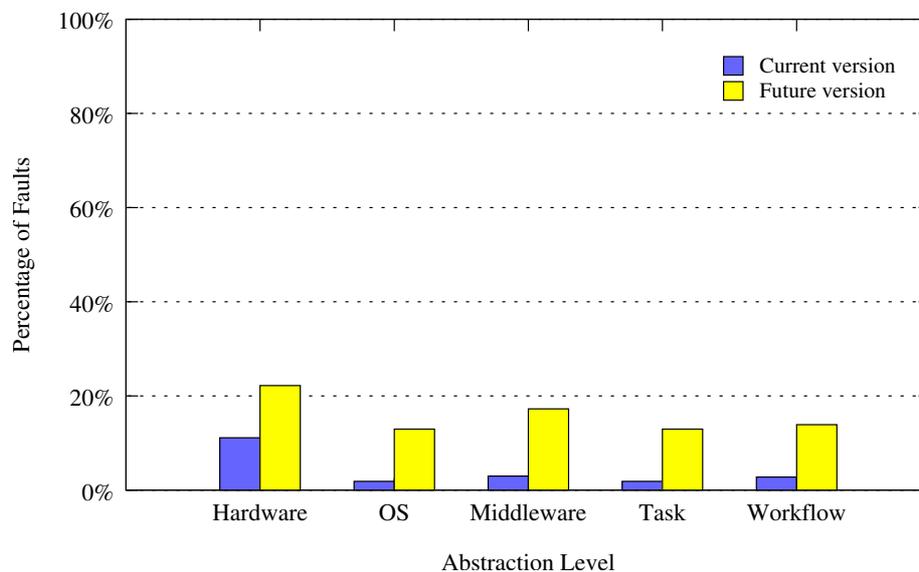


Figure 3: Average fault prevention

Figure 3 shows the percentage of the faults of every category that is prevented by a Grid workflow system on

average. We can see that fault prevention is virtually non-existent in the current versions of the studied Grid workflow systems, which is what we expected. We believe that the reason for this is to be found in the fact that Grid workflow management systems are usually working on a layer on top of the Grid middleware like Globus or Unicore, where the needed functionality to prevent such faults is not accessible to the systems.

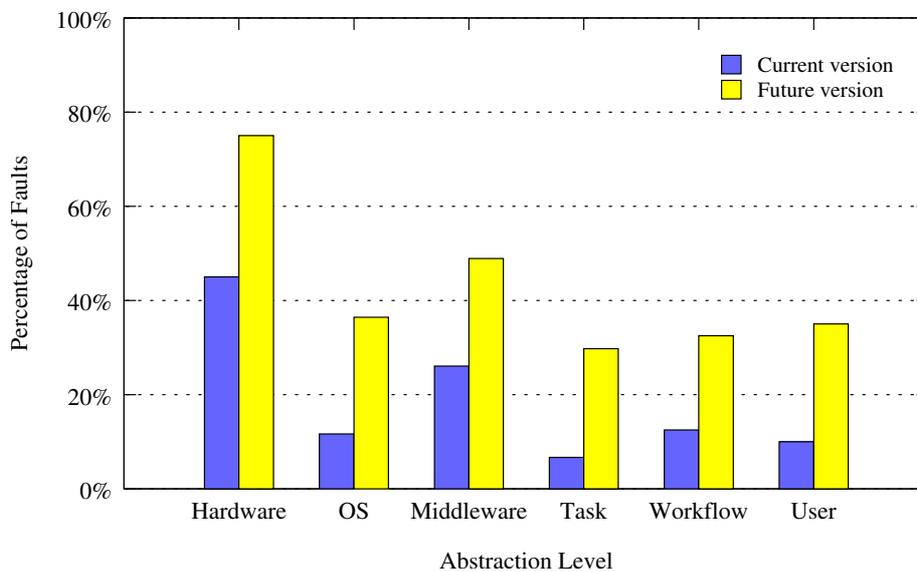


Figure 4: Average fault recovery

Figure 4 shows the percentage of the faults of every category that a Grid workflow system can recover from, on average. While the average Grid workflow system can recover from 45% of the hardware-level faults and 26% of the middleware-level faults, it can only recover from 11.6% of the OS-level faults, 12.5% of the Workflow-level faults and 6.7% of the Task-level faults. Taking a closer look at the task-level faults reveals that only four out of ten systems can recover from a job crash.

Generally, it can be said that current systems can recover from far fewer faults than they can detect, especially on the middleware and workflow-levels.

Figure 5 shows the percentage of systems that implement at least one of the fault tolerance techniques in each of the categories Task Redundancy (workflow-level redundancy and task-level redundancy techniques), Resubmission (task-level resubmission to the same resource, task-level resubmission to another resource), Workflow-level checkpointing (workflow-level light-weight and heavy-weight checkpoint/restart techniques), Task-level checkpointing (task-level checkpoint/restart and OS-level checkpoint/restart techniques), Management Service Redundancy (Workflow-level: choose another workflow management service, restart a workflow management service; Task-level: choose another task management service, restart a task management service) and User-defined exception handling.

As expected, the techniques that are the easiest to implement are used by most of the systems. Resubmission techniques are used by 80% of the systems; redundancy techniques are used by 40% of the current systems and are planned for implementation in another 30% of the systems in a future version, raising the support to 70% of the systems. While 60% of the systems use workflow-level checkpointing techniques, only 30% of the systems are using task-level checkpointing. This shows that it is still hard to implement task-level checkpointing in a satisfying way. While only 40% of the systems use management service redundancy techniques, this might be due to the fact that not all of the systems use a design that enables them to use redundant instances of management services. Surprisingly, only 20% of the systems enable users to define their own exception handling behavior.

## 5 Conclusions

This paper clearly shows that there definitely is an effort to make current workflow managers fault-tolerant. In the previous summary section several diagrams revealed the generally supported features and the open issues. As a final conclusion we cannot be satisfied with current achievements. Though the fault detection mechanisms are widely used,

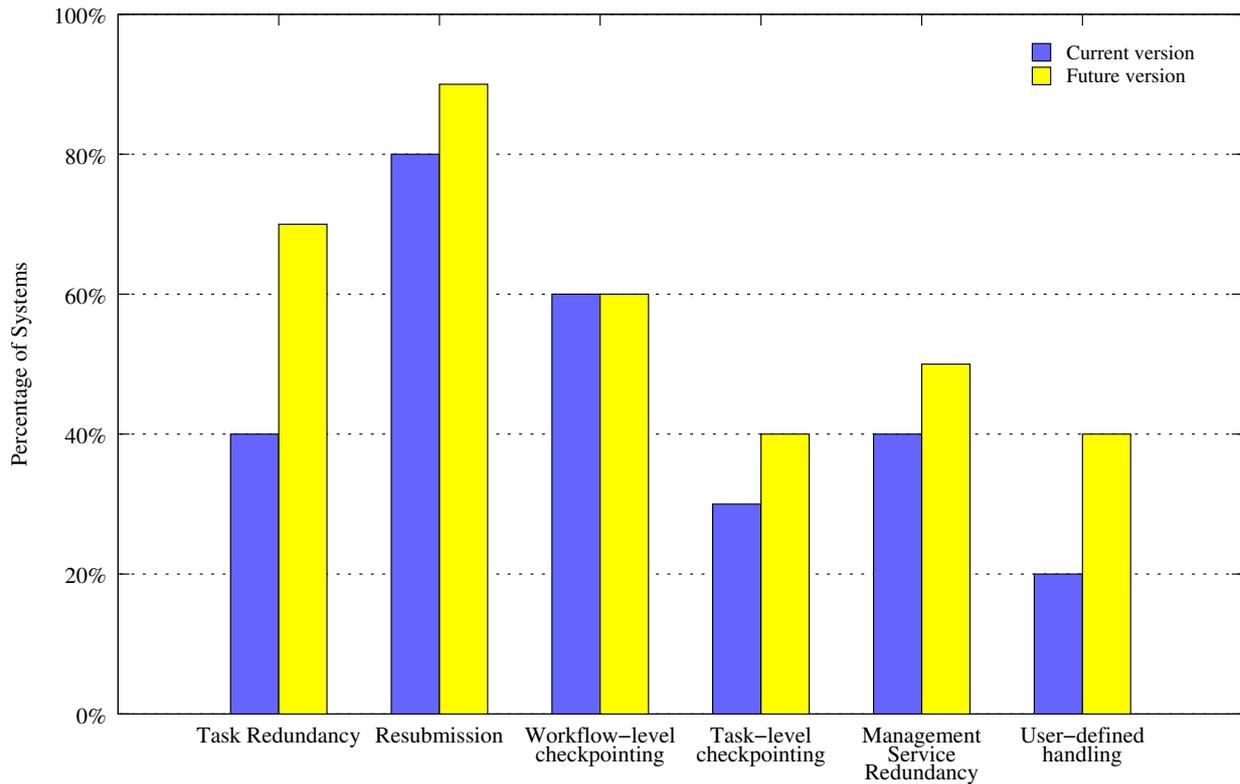


Figure 5: Percentage of systems that implement at least one fault tolerance technique of the specified category

current middleware limitations definitely means a border that the available systems cannot cross. New mechanisms should be developed to extend detection to lower levels, such as hardware and job execution faults. The prevention and recovery features are even weaker. Many of the detected faults are not handled with recovery, only little support is given to the users. Since grid development is moving towards creating self-aware solutions, these techniques need to appear in workflow enactors, too. We believe the current situation revealed in this paper helps researchers to focus on unsupported requirements, in this way future planning and work can be carried out more efficiently, paying more attention to user needs and system endurance.

## 6 Acknowledgement

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). We also would like to thank for the provided information to all the involved researchers of the overviewed systems.

## References

- [1] S. Hwang and C. Kesselman, Grid Workflow: A Flexible Failure Handling Framework for the Grid, in 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03), Seattle, Washington, USA, IEEE CS, Los Alamitos, CA, USA, June 22-23, 2003.
- [2] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, M. Wiczorek ASKALON: A Grid Application Development and Computing Environment, 6th International Workshop on Grid Computing, (C) IEEE Computer Society Press, November 2005, Seattle, USA

- [3] J. Yu and R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, *Journal of Grid Computing*, Volume 3, Numbers 3-4, Pages: 171-200, Springer Science+Business Media B.V., New York, USA, Sept. 2005.
- [4] P. Kacsuk, G. Sipos, Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal, *Journal of Grid Computing*, Feb 2006, pp. 1-18.
- [5] I. Foster C. Kesselman, The Globus project: A status report, in *Proc. of the Heterogeneous Computing Workshop*, IEEE Computer Society Press, 1998, pp. 4-18.
- [6] <http://www.globus.org/toolkit/docs/4.0/>
- [7] <http://glite.web.cern.ch/glite/>
- [8] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bala, E. del Grosso, M. Casalegno, N. Piclin, M. Pintore, W. Sudholt and K. Baldrige, Chemomentum - UNICORE 6 based infrastructure for complex applications in science and technology, *Proceedings of 3rd UNICORE Summit 2007 in conjunction with EuroPar 2007*, Rennes, France
- [9] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder, UNICORE - From Project Results to Production Grids, L. Grandinetti (Edt.), *Grid Computing: The New Frontiers of High Performance Processing*, *Advances in Parallel Computing 14*, Elsevier, 2005, pages 357-376
- [10] D. Laforenza, R. Lombardo, M. Scarpellini, M. Serrano, F. Silvestri, P. Faccioli, Biological Experiments on the Grid: A Novel Workflow Management Platform, pp. 489-494, *Twentieth IEEE International Symposium on Computer-Based Medical Systems (CBMS'07)*, 2007
- [11] <http://www.activebpel.org>
- [12] E. Elmroth, F. Hernandez and J. Tordsson. A light-weight Grid workflow execution service enabling client and middleware independence. *Proceedings of the Grid Applications and Middleware Workshop, PPAM 2007*, Springer-Verlag, Lecture Notes in Computer Science.
- [13] A. Hoheisel and M. Alt: Petri Nets. In: *Workflows for eScience*, Ian J. Taylor, D. Gannon, E. Deelman, and M. S. Shields (Eds.), Springer, 2006
- [14] <http://www.kwfgrid.eu/>
- [15] <http://www.w3.org/TR/soap/>
- [16] <http://www.ebay.com/>
- [17] N. Mandal, E. Deelman, G. Mehta, M-H. Su, and K. Vahi, Integrating Existing Scientific Workflow Systems: The Kepler/Pegasus Example, *Proceedings of the Second Workshop on Workflows in Support of Large-Scale Science (WORKS'07)*, in conjunction with the IEEE International Symposium on High Performance Distributed Computing Monterrey, CA, June 2007
- [18] D. Thain, T. Tannenbaum, and M. Livny, *Distributed Computing in Practice: The Condor Experience*, *Concurrency and Computation: Practice and Experience*, 2005, pp. 323-356.
- [19] F. Baude, D. Caromel, C. Delbe and Ludovic Henrio, A Hybrid Message Logging-CIC Protocol for Constrained Checkpointability, *Proceedings of EuroPar 2005*, September 2005, Lisbon, Portugal
- [20] I. Taylor, M. Shields, I. Wang, and A. Harrison, The Triana Workflow Environment: Architecture and Applications, In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 320-339. Springer, New York, Secaucus, NJ, USA, 2007.
- [21] G. Allen et. al., Enabling Applications on the Grid: A GridLab Overview, *International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications*, August 2003

## Appendix: The Questionnaire

### CoreGrid Work Package 5 - Grid Information and Monitoring Services Questionnaire about fault-tolerance support in Grid workflow systems

Institute of Computer Science, University of Innsbruck. Austria  
MTA SZTAKI, Budapest, Hungary

**Contact:**  
Kassian Plankensteiner  
kassian.plankensteiner@dps.uibk.ac.at

#### Definition

In this document, a *workflow* is a set/graph of *tasks* connected using data- and/or control-flow edges.

#### General Questions

Name of your Grid Workflow system:                    insert system name

Website with information about the system:    insert URL

Is there any support for fault-tolerance in the current version of your Grid workflow system?

Yes             No

If yes, since when does your system support fault-tolerance techniques (e.g. since 2 years)?

If not, have you planned to implement fault-tolerance techniques for a future version?

Yes             No, because insert reason

On a scale from 1 (very important) to 6 (not important), how important is fault-tolerance for the future of your system?

(very important) 1   2   3   4   5   6 (not important)

## Fault detection

Can your system detect and cope with the following faults? (add your own in the additional rows)

Fault type	Fault description	System can detect		System can prevent		System can recover from	
		Current version	Future version	Current version	Future version	Current version	Future version
Hardware	Machine crashed / down	<input type="checkbox"/>					
	Network down	<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					
Operating System	Disk quota exceeded	<input type="checkbox"/>					
	Out of memory	<input type="checkbox"/>					
	Out of disk space	<input type="checkbox"/>					
	File not found	<input type="checkbox"/>					
	Network congestion	<input type="checkbox"/>					
	CPU time limit exceeded	<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					
Middleware	Authentication failed	<input type="checkbox"/>					
	Job submission failed	<input type="checkbox"/>					
	Job hanging in the queue of the local resource manager	<input type="checkbox"/>					
	Job lost before reaching the local resource manager	<input type="checkbox"/>					
	Too many concurrent requests	<input type="checkbox"/>					
	Service not reachable / not responding	<input type="checkbox"/>					
	File staging failure	<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					
Task	Memory leak	<input type="checkbox"/>					
	Uncaught Exceptions (e.g. numerical)	<input type="checkbox"/>					
	Deadlock / Livelock	<input type="checkbox"/>					
	Incorrect output data	<input type="checkbox"/>					
	Missing shared libraries	<input type="checkbox"/>					
	Job crashed	<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					
Workflow	Infinite Loop	<input type="checkbox"/>					
	Input data not available	<input type="checkbox"/>					
	Input error	<input type="checkbox"/>					
	Data movement failed	<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					

User	User-definable exceptions <sup>1</sup>	<input type="checkbox"/>					
	User-definable assertions	<input type="checkbox"/>					
		<input type="checkbox"/>					
		<input type="checkbox"/>					

Comments and additional explanation (optional):

---

<sup>1</sup> *User-definable exception*: The user defines a condition for an exception (e.g. if the output data size is bigger than 5 MB, a fault happened)  
 CoreGRID TR-0091

Which service of your system is capable of detecting the following faults? (e.g. Enactor, Scheduler, Broker, ...) If you are using a third-party provided software for detection, please name that instead of a service of your system.

Fault type	Fault description	Name of service
Hardware	Machine crashed / down	
	Network down	
Operating System	Disk quota exceeded	
	Out of memory	
	Out of disk space	
	File not found	
	Network congestion	
	CPU time limit exceeded	
Middleware	Authentication failed	
	Job submission failed	
	Job hanging in the queue of the local resource manager	
	Job lost before reaching the local resource manager	
	Too many concurrent requests	
	Service not reachable / not responding	
	File staging failure	
Task	Memory leak	
	Uncaught Exceptions (e.g. numerical)	
	Deadlock / Livelock	
	Incorrect output data	
	Missing shared libraries	
	Job crashed	
Workflow	Infinite Loop	
	Input data not available	
	Input error	
	Data movement failed	
User	User-definable exceptions (explain in a footnote)	
	User-definable assertions	

Comments and additional explanation (optional):

## Fault Prevention and Recovery

(After the fault happened)

Which fault recovery/prevention mechanisms does your system support?

Abstraction Level	Description	System supports	
		Current version	Future version
Workflow	Redundancy	<input type="checkbox"/>	<input type="checkbox"/>
	Light-weight <sup>2</sup> checkpoint/restart mechanism	<input type="checkbox"/>	<input type="checkbox"/>
	Heavy-weight <sup>3</sup> checkpoint/restart mechanism	<input type="checkbox"/>	<input type="checkbox"/>
	Transaction / Rollback mechanism	<input type="checkbox"/>	<input type="checkbox"/>
	Workflow replication	<input type="checkbox"/>	<input type="checkbox"/>
	Data replication	<input type="checkbox"/>	<input type="checkbox"/>
	Choose another workflow management service	<input type="checkbox"/>	<input type="checkbox"/>
	Restart a workflow management service	<input type="checkbox"/>	<input type="checkbox"/>
	Task management service reliability measurement	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
Task	Retry / Resubmission to the same resource	<input type="checkbox"/>	<input type="checkbox"/>
	Resubmission to another Resource	<input type="checkbox"/>	<input type="checkbox"/>
	Task Migration	<input type="checkbox"/>	<input type="checkbox"/>
	Checkpoint/restart mechanism	<input type="checkbox"/>	<input type="checkbox"/>
	Checkpoint/restart on the OS level	<input type="checkbox"/>	<input type="checkbox"/>
	Task Replication <sup>4</sup>	<input type="checkbox"/>	<input type="checkbox"/>
	Alternate Task <sup>5</sup>	<input type="checkbox"/>	<input type="checkbox"/>
	Choose another task management service	<input type="checkbox"/>	<input type="checkbox"/>
	Restart a task management service	<input type="checkbox"/>	<input type="checkbox"/>
	Resource reliability measurement	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
User	User-defined exception handling	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>

Comments and additional explanation (optional, e.g. how does your system prevent faults and which can it prevent, how do your fault-recovery techniques work, ...):

<sup>2</sup> *Light-Weight Checkpointing*: saving only the current location of the intermediate data, not the data itself. Fast, but restarting can only work as long as the intermediate data is available at its original location

<sup>3</sup> *Heavy-Weight Checkpointing*: saving all of the intermediate data in a place where it can be kept for as long as needed.

<sup>4</sup> *Task Replication*: create multiple alternative Task implementations simultaneously

<sup>5</sup> *Alternate Task*: using a different Task implementation if one fails

Which service of your system provides the following fault-recovery techniques? (e.g. Enactor, Scheduler, Broker, ...) If you are using a third-party provided software for recovery, please name that instead of a service of your system.

Abstraction Level	Description	Name of service
Workflow	Redundancy	
	Light-weight checkpoint/restart mechanism	
	Heavy-weight checkpoint/restart mechanism	
	Transaction / Rollback mechanism	
	Workflow replication	
	Data replication	
	Choose another workflow management service	
	Restart a workflow management service	
	Task management service reliability measurement	
Task	Retry / Resubmission to the same resource	
	Resubmission to another Resource	
	Task Migration	
	Checkpoint/restart mechanism	
	Checkpoint/restart on the OS level	
	Task Replication	
	Alternate Task	
	Choose another task management service	
	Restart a task management service	
	Resource reliability measurement	
User	User-defined exception handling	

Comments and additional explanation (optional):

Please provide additional comments, hints and references about fault-tolerance for Grid workflow systems: