

## Scalable multilevel checkpointing for distributed applications - on the possibility of integrating Total Checkpoint and AltixC/R

*Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolajczak*  
{gracjan, radekj, rafal.mikolajczak}@man.poznan.pl  
*Poznan Supercomputing and Networking Center*  
*61-704 Poznan, Noskowskiego 12/14, Poland*

*Jozsef Kovacs*  
smith@sztaki.hu  
*Computer and Automation Research Institute of Hungarian Academy of Sciences*  
*1111 Budapest Kende u. 13-17. Hungary*



CoreGRID Technical Report  
Number TR-0035  
June 1, 2006

Institute on Grid Information, Resource and Workflow  
Monitoring Services

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

# Scalable multilevel checkpointing for distributed applications - on the possibility of integrating Total Checkpoint and AltixC/R

Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolajczak  
{gracjan, radekj, rafal.mikolajczak}@man.poznan.pl  
Poznan Supercomputing and Networking Center  
61-704 Poznan, Noskowskiego 12/14, Poland

Jozsef Kovacs  
smith@sztaki.hu  
Computer and Automation Research Institute of Hungarian Academy of Sciences  
1111 Budapest Kende u. 13-17. Hungary

*CoreGRID TR-0035*

June 1, 2006

## Abstract

This is the second document in a series of reports describing the work that aims to integrate the SZTAKI's high-level checkpointing tool Total Checkpoint (TCKPT) with one of PSNC's low-level checkpointers. The low-level checkpointer considered in this report is AltixC/R and the high-level checkpointer remains the TCKPT. As the functionality of the tandem is scaled according to the functionality of the low-level checkpointer, the final mechanism would inherit the benefits of the AltixC/R checkpointer. On the basis of the knowledge and experience gained with help of the previous report and the present one, we will finally take a decision on which PSNC's checkpointing package will be integrated with the TCKPT.

## 1 Introduction

This report has been performed within the CoreGRID project by the Research Group (RG) working on Checkpointing Services. The long-term goal of that RG is to define the Grid Checkpointing Architecture (GCA) [6] [7], while the short-term goal is to integrate one of PSNC's checkpointing packages [10] [11] with SZTAKI's Total Checkpoint (TCKPT), which will allow utilizing the PSNC's checkpointing package to checkpoint and recover PVM applications [14] [15]. However, there is one important obstacle that makes the integration difficult. The PSNC's checkpointing packages and SZTAKI's TCKPT were designed and implemented for different platforms. Additionally the TCKPT requires a peculiar interface from the checkpointing package that it cooperates with. Then to integrate the TCKPT with one of PSNC's checkpointing packages, some porting and adapting activity is necessary. The possibility of porting and adapting the psncLibCkpt [2] (one of PSNC's checkpointing packages) to requirements imposed by TCKPT was scrutinized in [1]. This report is considered as a continuation of work presented in the [1] and goes into an analysis of the possibility and labor that should be done in order to make AltixC/R [4] [5] ready to cooperate with TCKPT.

In the next sections of this report there are outlines of the involved products (AltixC/R and TCKPT) and a more detailed description of the interface imposed by TCKPT as well as of the possibility of integrating it with AltixC/R. Additionally, an estimated schedule of the actual adapting AltixC/R to TCKPT is presented.

---

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

## 2 AltixC/R outline

The AltixC/R is the kernel-level checkpointing package designed for Altix [18] systems equipped with IA64 [17] processors and running under the ProPack [16] environment (a Linux-based environment prepared by SGI). Versions of the package that works with ProPack based on Linux kernel 2.4 as well as with a more recent ProPack that is based on Linux kernel 2.6 [19] [20] are currently available. The package is characterized by all features typical for kernel-level approach. It is easy to use, there is no assumption on the availability of source codes or the programming language that was used to write the programs that are to be checkpointed. The package has to be deployed by the system administrator. The package allows doing checkpoints of multi-process programs that communicate through System V IPC objects. Additionally, the idea of virtualization of some system global keys and identifiers has been employed in that product. Thanks to that, when the program is recovered, it is cheated that the identifiers have not changed [8] (even though, due to technological reasons, it is very likely that they have). The package has been developed as part of the SGIGrid project [9].

The workflow of checkpointing and recovery activity is essential from the point of view of integration. Therefore the way the checkpoint is taken and the process is recovered by AltixC/R is presented below.

### 2.1 Scenario of taking checkpoint with AltixC/R

The AltixC/R allows taking checkpoints of any processes that adhere to imposed limitations, i.e. the processes that use only the resources that are supported by AltixC/R. It is not important in what language the process that is being checkpointed has been written. The checkpointed process is totally independent of the checkpointing tools. It means that the process does not have any hooks or other common elements with the AltixC/R package. Actually the being checkpointed process is completely unaware of the existence of the checkpointing tool.

The tools, resources and sequence of the steps involved in the process of taking the checkpoint are presented in figure 1. When the user decides to take checkpoint, he/she executes the "chkpnt <PID>" command (where <PID> specifies the process being checkpointed). With help of the system calls, the procfs filesystem and special proprietary kernel modules (*syscover* and *ckpt* modules) the chkpnt command saves the image of the process defined by PID into the file. What is important, the whole operation is transparent to the process being checkpointed and there is no way to force the process to perform some special procedures before and just after the checkpoint is done. It means that the being checkpointed process releases and reallocates non-checkpointable resources when necessary.

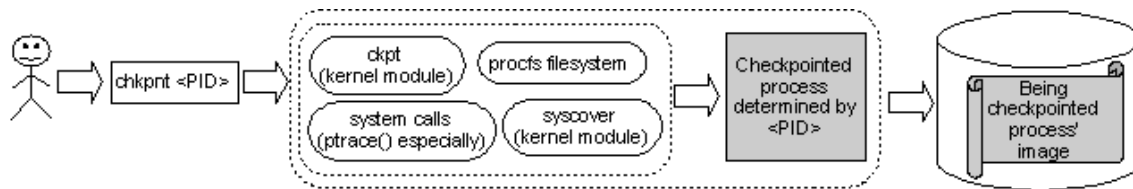


Figure 1: Taking checkpoint with AltixC/R

### 2.2 Scenario of recovering process with AltixC/R

The recovering procedure is also completely transparent to the process being recovered. The resources, tools and the sequence of operations involved in the recovering process are presented in figure 2. In order to recover the process, the user has to execute the "resume <image>" command, where <image> is the path to the image of the checkpointed process. To obtain the "frame" of the process being recovered, the resume command "forks" itself. Further, using the available system calls, the procfs filesystem and proprietary kernel modules the resume command replaces the memory space and other resources of the just forked process with those saved in the checkpoint image. Finally, the resume command finishes execution and the just recovered process begins to be executed from the point where the checkpoint was taken. The recovered process has no possibility to interact with the recovery command, so registering and invoking of any callbacks is not possible (from TCKPT point of view it is very important remark).

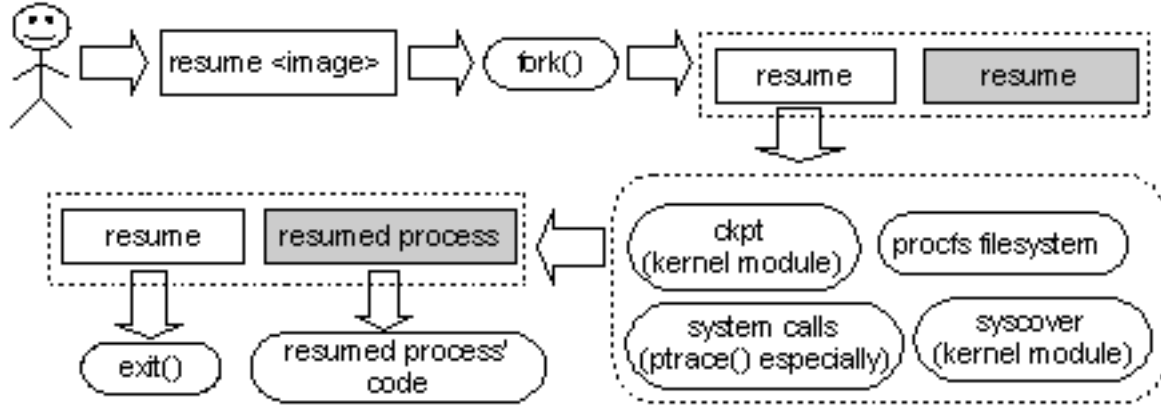


Figure 2: Process recovering with AltixC/R package

### 3 TCKPT outline

TotalCheckpoint (TCKPT) is a high-level checkpointing environment allowing taking checkpoints of parallel, PVM-based applications. The environment is meant for the IA32 platform with Linux kernel 2.4. Under the hood it employs a low-level checkpointing library. At the moment, the low-level checkpointing library is the *Ckpt library* [21], developed at the University of Wisconsin (but we strive for providing the aforementioned *psncLibCkpt* library or AltixC/R as the alternate low-level checkpointing).

TCKPT requires setting up a coordination process (one per cluster or site) running as a daemon in the background and re-linking of the user application. Files used by the users application are also saved, and as a result of a checkpointing action, an application description (XML file) is also produced to let the upper layers know about the internals of the application, for example, a list of working files. Through this description the renaming of the working files for a migrating application and the site migration is also supported.

The TCKPT tool is prototyped and introduced to support the Hungarian nationwide ClusterGrid infrastructure maintained by the National Information Infrastructure Development Office. ClusterGrid is built by PC clusters provided by academic institutes and universities.

We found the differences in assumptions about the way the checkpoint is taken and the process is recovered the most bothered issue on the road to integrate the TCKPT and AltixC/R. Therefore, to help understand the conclusions and propositions presented in this report, the way the checkpoint is taken and the process is recovered within the TCKPT environment is presented below.

#### 3.1 Scenario of taking checkpoint within TCKPT environment

First, it is important to realize that TCKPT is meant for being used with PVM applications and third-party low-level checkpointing library. Then the first inconsistency with AltixC/R is that it is not a library but a kernel-level mechanism. In the TCKPT environment an individual process constituting the PVM application is built of a few layers. On the highest one there is the code of PVM application itself, i.e. a code performing some parallel computation. This code utilizes the library (PVM lib layer) and services provided by the PVM environment. The next layer is the TCKPT library that manages checkpointing-related affairs, intercepts PVM related functions and mediates in calling actual PVM library functions. That layer cooperates with the third-party checkpointing library which stores the image of the process to persistent memory. All these layers are merged into one process and reside in the same memory space. There is an assumption that the checkpointing library exports functions that allow to take the checkpoint of the current process and to restore the process to the state from the time when the checkpoint was being taken. Additionally there is an assumption that the TCKPT lib can register callback functions with the checkpointing library. These callbacks are called by the checkpointing library just before and just after the checkpoint is taken and just after the memory space of PVM application is recovered, respectively. The flow of control between individual layers is depicted in figure 3.

The checkpointing activity in the TCKPT environment looks as follows. The TCKPT library determines that checkpoint has to be taken (it is not important for us how it made this decision) and relinquishes the resources that the checkpointing library is not able to support properly. There is an assumption that the checkpointing library cannot deal

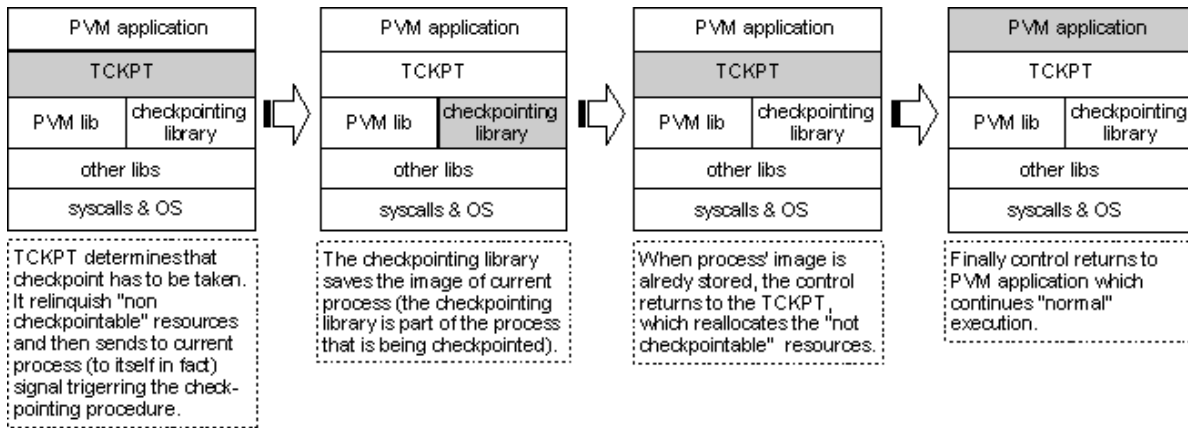


Figure 3: Workflow of checkpointing activity

with PVM-related resources so this is why TCKPT takes care of them. When the unsupported resources are already unallocated, the TCKPT asks the checkpointing library to store memory and other crucial resources (an accurate set of supported resources depends on a particular checkpointing library). When the checkpointing library finishes its job, then the control returns to the TCKPT which reallocates the "non-checkpointable resources". Finally the control returns to the PVM application which continues the "normal" execution.

### 3.2 Scenario of recovering process within TCKPT environment

During the recovery stage, the TCKPT environment also takes some assumptions on the flow of control and on the way the individual process is restored. The flow of control between layers constituting the process of the PVM job is depicted in figure 4. When the checkpointing library finishes restoring the memory space of the involved process, then it calls a special callback function registered by TCKPT. This callback performs some actions required to proper recovering of the PVM environment. Finally, when the callbacks finishes, the control is returned to the PVM application to the place where the checkpoint was taken.

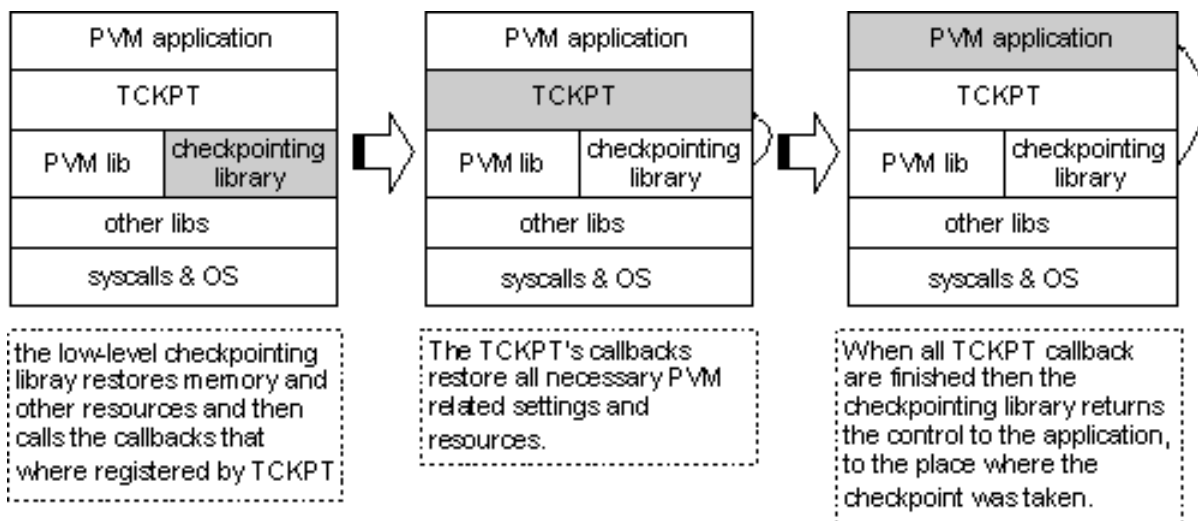


Figure 4: Recovering process in TCKPT environment

It is noticeable that in case of AltixC/R the flow of control presented in figure 4 can not be achieved directly. No part of Altix C/R is placed in the address space of the process being checkpointed or restarted. Additionally the procedure of taking checkpoint or recovering the process is atomic in a sense that there is no way to call any

callbacks after the checkpoint is taken but before the control returns to the PVM application (besides the procedure of the process' recovering is also atomic).

## 4 Providing AltixC/R with interface imposed by TCKPT

It results from the above description of semantics of the TCKPT and AltixC/R product that the biggest problem on the road to integrate these products is to cheat the TCKPT that it deals with the checkpointing library (and not a kernel-level checkpointing package) and to figure out the way the TCKPT could register appropriate callbacks. The analysis of those and other integration-related issues is presented below.

### 4.1 Adapting the AltixC/R's commands to the TCKPT requirements

The first idea to fit AltixC/R to TCKPT is to write a checkpointing library which in fact would be the wrapper to the actual commands. It is depicted in figure 5a. Such solution would be quite simple and fast to implement but unfortunately it is not acceptable due to the atomic character of the `chkpnt` and `resume` tools. For example, when the `resume` tool is invoked, then when it finishes, the target process is considered as restored and the control is passed to it (there is no way to call any callback just after the memory is restored but before the control is returned to the restored process).

Because of that we figured out another solution. As it is depicted in figure 5b, we proposed that the AltixC/R's commands (`chkpnt` and `resume`) are completely embedded into the checkpointing library. It means that the source code of these commands has to be slightly redesigned and compiled as the library which will later be linked with PVM programs. Unfortunately, after a more accurate analysis it turned out that the AltixC/R package does not allow for `chkpnt` and `resume` commands to operate on the current process (both `chkpnt` and `resume` commands are allowed to operate on external processes). Luckily, after further research effort we discovered that the issue can be overcome by wise redesigning of the AltixC/R's kernel level modules.

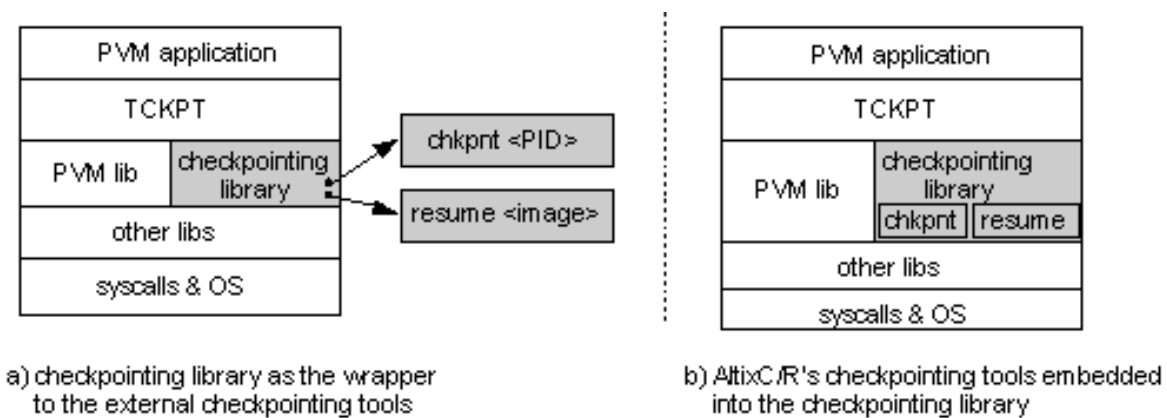


Figure 5: Adapting AltixC/R's commands to the TCKPT requirements

The example workflow of the checkpointing activity within the TCKPT environment integrated with AltixC/R is depicted in figure 6. The description of successive steps was omitted but it is analogous to those from figure 3.

The adequate figure presenting the workflow of the recovering procedure would be created in a similar way from figure 4.

From now on, there is an assumption that the phrase "AltixC/R checkpointing library" refers to the potential library that implements the interface required by TCKPT and that has embedded the functionality of the AltixC/R's commands (`chkpnt` and `resume` commands).

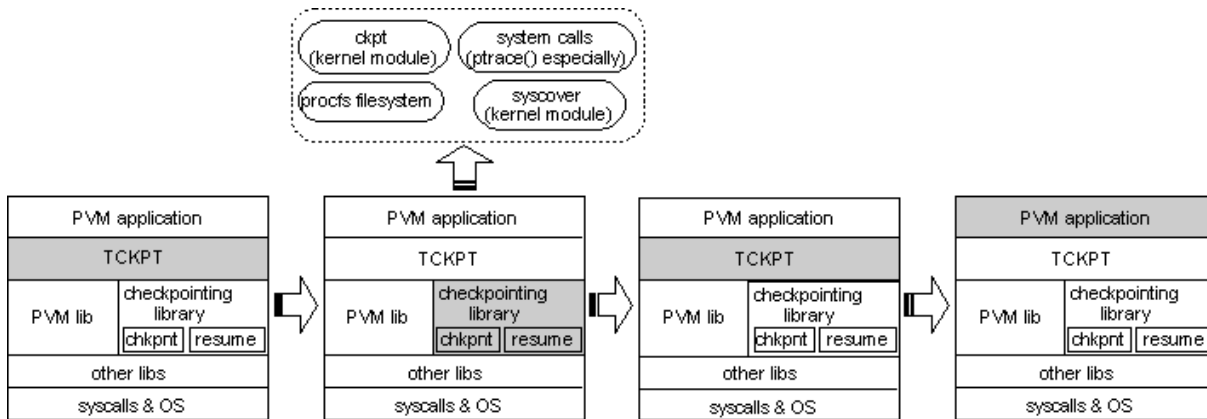


Figure 6: Workflow of checkpointing activity (TCKPT with AltixC/R)

## 4.2 Restoring open files

As it was described in the [1] in section 5.1, the TCKPT environment imposes the peculiar semantics of restoring files. TCKPT expects that the checkpointing package allows registering callbacks and that the checkpointing package exports a special function *ckpt\_fds()* which can be called from within the callback. The *ckpt\_fds()* function is used by TCKPT to adjust the way the open files are restored (a more accurate description of that mechanism can be found in the [1]). Currently the AltixC/R possesses its own mechanism of restoring open files which is not compatible with TCKPT. Providing that AltixC/R checkpointing library adheres to the architecture depicted in figure 5b, adapting that mechanism to TCKPT requirements would be possible.

## 4.3 Naming the checkpoint images and their format

Before TCKPT submits the checkpoint request to its low-level checkpointer, it first has to set the name of the outcome checkpoint image. Additionally, TCKPT supposes that the image consists of one file that contains all checkpoint-related data. Unfortunately, the AltixC/R does not allow to control the name of the image file at run-time and the image in fact consists of a set of files.

The solution about the possibility of naming checkpoint images is described in section 4.7. The problem concerning the quantity of files that constitute the whole image will be solved by redesigning the TCKPT. It will be able to deal with multi-file images by means of the base-name concept. The base-name will represent the prefix that will be used to construct all involved file names.

## 4.4 Continuing or ending process after checkpoint

The TCKPT expects to have the possibility to finish the process after checkpoint is taken. Currently the AltixC/R does not meet these requirements but from the technical point of view it is simple to add such functionality.

## 4.5 Signal triggering the checkpoint

The TCKPT assumes that the checkpoint is triggered by signal and additionally that it is possible to set which signal will be used to do that. Currently the AltixC/R does not adhere to that model but it is quite simple to provide the AltixC/R checkpointing library with a special signal handler that would be the entry point to the checkpoint procedure. However, a more complex issue is how to register the handler and how to associate it with a particular signal. As we do not plan to equip the AltixC/R checkpointing library with some mechanism intercepting the control at startup time, the setup functions would have to be called explicitly. Then we take an assumption that in a prototype design the only considered mechanism for that purposes is exporting the *ckpt\_config()* function. The solution is described in more detail in section 4.7.

## 4.6 Registering callback functions

The TCKPT has to register with the checkpointing library some callback functions that are called just before and just after checkpoint is taken and just after the process is recovered but before the control returns to it. TCKPT expects that in order to allow registering the callbacks, the checkpointing library exports the following functions:

```
void ckpt_on_preckpt(void (*f)(void *), void *arg); - to register the function called just before checkpoint is taken
```

```
void ckpt_on_postckpt(void (*f)(void *), void *arg); - to register the function called just after checkpoint is taken
```

```
void ckpt_on_restart(void (*f)(void *), void *arg); - to register the function called just after the process is recovered but before control is returned to the original program. In fact, when the callback function finishes, then before control is returned to the program, the open files have to be restored by the checkpointing library.
```

Then these functions have to be implemented by the AltixC/R library, and the registered callbacks have to be invoked in adequate time.

## 4.7 The possibility of programmable configuration of checkpointing library

The TCKPT requires the possibility of adjusting some checkpointing library properties at run-time. It expects that the checkpointing library allows to configure itself at run-time by means of the function of the following signature (this function has to be exported by the checkpointing library):

```
void ckpt_config(struct ckptconfig *cfg, struct ckptconfig *old);
```

The passed arguments, *cfg* and *old*, serve to pass new configure options and to store the old one, respectively.

The definition of the structure that the arguments adhere to is as follows:

```
struct ckptconfig {  
    /* For users */  
    char name[CKPT_MAXNAME];  
    unsigned int asyncsig;  
    unsigned int continues;  
    unsigned int msperiod;  
    int flags;  
};
```

With help of the fields of the above structure, the TCKPT can adjust such checkpointer's parameters as: the name of image, the triggering signal, the behavior after checkpoint and others. An exact description is out of this paper. It seems that SCLL could accept and properly handle only the fields important from the point of view of TCKPT.

## 4.8 Support for pipes

The TCKPT takes advantage of pipes for internal purposes but releases them just before the process image is saved (it is done in one of callback functions). However, the matter of support for pipes is not quite clear. The low-level checkpointer that the TCKPT is using at the moment has support for pipes and we were not able to make an experiment if the whole environment works properly without support for pipes. In general it seems that the low-level checkpointer does not have to support the pipes but (in opposite to the psncLibCkpt described in [1]) the AltixC/R supports the pipes as well, so even if that support was necessary, the AltixC/R suits that requirement.

Lp.	Task description	Duration
1.	Redesign and reimplementation of the "ckpt" dynamic kernel module in order to allow it to cooperate with applications that want to "checkpoint themselves" (according to semantics depicted in figure 3).	half a week
2.	Redesign and reimplementation of the "ckpt" dynamic kernel module in order to allow it to cooperate with applications that want to "recover themselves" (according to semantics depicted in figure 4).	half a week
3.	Moving the functionality of the AltixC/R's ckpt command to the AltixC/R checkpointing library. This task entails a little redesign of the internal ckpt's algorithm.	one and half a week
4.	Moving the functionality of the AltixC/R's resume command to the AltixC/R checkpointing library. This task (as the previous one) entails a little redesign of the internal resume's algorithm.	one week
5.	Providing AltixC/R with entry points to checkpoint/restart functionality required by TCKPT (i.e. ckpt_restart() function and interface to the triggering checkpoint see section 4.5).	one week
6.	Adding to the AltixC/R checkpointing library the possibility of registering the required callbacks and mechanism for calling them in appropriate time.	one weeks
7.	Adding support for the required semantics of restoring files. The semantics is mentioned in section 4.2 and described in more detail in [1].	one and half a week
8.	Providing the AltixC/R checkpointing library with the possibility of programmable configuration (see section 4.7).	two weeks
9.	Other minor integration works (e.g. adding a mechanism allowing to terminate the process just after checkpoint is taken).	one week

Table 1: Integration activity schedule.

## 4.9 Dynamic vs. static library

The potential AltixC/R checkpointing library can be implemented as a static or dynamic library. The TCKPT can cooperate with both.

## 4.10 Cooperation with checkpointing server

The TCKPT environment introduces the checkpointer server which serves as a checkpoint images repository. The low-level checkpointer has to be able to cooperate with this server. However, in the presence of the NFS, the TCKPT environment can work without the checkpointing server. Then, the first prototype implementation of the AltixC/R checkpointing library would miss the support for the checkpointing server.

## 4.11 Settling the flow of the control path

As it was described in [1], the integration of TCKPT with the psncLibCkpt checkpointing library would require reconciling the flow of the control path of the checkpointing library and the TCKPT library. In case of AltixC/R, the tool itself does not intercept any system calls or other libc library functions. Additionally, the AltixC/R library is not even planned to intercept the control in order to perform some startup operations (the TCKPT would have to explicitly call the setup function of the AltixC/R library). Hence the assumed flow of the control of the AltixC/R checkpointing library does not disturb the TCKPT library.

# 5 Integration schedule

In this section we are going to provide a draft schedule of work required to adapt the AltixC/R to interface required by TCKPT. According to the analysis performed in section 4, the work can be divided in three general stages:

1. Redesign and reimplementation of AltixC/R's kernel-level modules so that they could be used in the checkpointing and recovering model described in section 4.1.

2. Moving the functionality of AltixC/R's user-level commands to the AltixC/R checkpointing library (according to considerations made in section 4.1).
3. Extending the checkpointing library (designed and implemented in the previous stage) with the interface required by TCKPT.
  - Providing the *ckpt\_fds()* function which will allow the TCKPT on adjusting the files recovering process (according to considerations made in section 4.2).
  - Extending the checkpointing package with the possibility of forcing self-termination after checkpoint is done (according to considerations made in section 4.4).
  - Extending the checkpointing package with the possibility of setting the signal that is to be used to trigger checkpoint (according to the section 4.5).
  - Implementing the possibility of registering callbacks related with specific events (according to requirements described in section 4.6).
  - Adding the possibility of run-time, programmable configuration of crucial features of the checkpointing package (see section 4.7).

Table 1 presents estimated time and tasks required to accomplish the adaptation of AltixC/R to the TCKPT.

## 6 Conclusions

From a logical point of view the TCKPT and AltixC/R are complementary products. However, in practice it has turned out that there are significant obstacles to integrate them seamlessly: different target platforms and a particular interface imposed by TCKPT. In spite of different target platforms of TCKPT and AltixC/R and despite some other difficulties we decided to check if it would be possible to integrate these two products. Additionally an interesting question is how much time-consuming the potential integration would be. Thanks to the analysis that we performed we have received the answer. The integration seems to be not trivial but possible. Moreover, in comparison to the [1] (a report about adapting *psncLibCkpt* to the TCKPT) the analysis results are a little surprising. Even though the AltixC/R seems not to fit the TCKPT requirements at all (it is not a library, while the TCKPT cooperates only with libraries), the estimated adaptation time is shorter than that required to the adaptation of the *psncLibCkpt*. It results from the fact that in case of *psncLibCkpt* it actually turned out that a large part of the code should be completely redesigned and reimplemented only to port the checkpointing functionality to the Linux OS. In case of AltixC/R we met some complex issues (the most crucial problem is described in section 4.1) but we proposed reasonable solutions for those issues and additionally we estimate to apply them in rational time. Moreover, implementation of a required AltixC/R checkpointing library should not be really time-consuming because to a large extent it boils to reusing the existing source codes of AltixC/R's user-level commands.

Taking into account the above conclusions and the fact that Altix and other IA64-based platforms are increasingly becoming more and more popular, we lean towards choosing the AltixC/R package as the one which will be used in the actual TCKPT and one of PSNCs checkpointing packages integration. However, the final decision will be taken after the analysis of a porting possibility of TCKPT to the Altix platform is finished.

Referring to the long-term goal of the RG that prepared the report (namely defining GCA), after the integration of TCKPT with one of PSNC's checkpointing packages, the result will be considered as the Core Service from the point of view of GCA [6] [7].

## References

- [1] Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolajczak and Jozsef Kovacs, Scalable multilevel checkpointing for distributed applications on the integration possibility of TCKPT and *psncLibCkpt*, CoreGRID TR0019.
- [2] <http://checkpointing.psnclib.pl/Progress/psncLibCkpt/>
- [3] <http://www.lpds.sztaki.hu/pgrade/>

- [4] <http://checkpointing.psnc.pl/SGIGrid/>
- [5] Gracjan Jankowski, Rafal Mikolajczak, Radoslaw Januszewski, Checkpoint/Restart mechanism for multiprocess applications implemented under SGIGrid Project, Proceedings of the Cracow Grid Workshop 2004, pp.142-149, ISBN: 83-911541-4-5, 2005.
- [6] Gracjan Jankowski, Jozsef Kovacs, Norbert Meyer, Radoslaw Januszewski and Rafal Mikolajczak, Towards Checkpointing Grid Architecture, To be published in PPAM2005 proceedings.
- [7] Gracjan Jankowski, Radoslaw Januszewski, Jozsef Kovacs, Norbert Meyer and Rafal Mikolajczak, Grid Checkpointing Architecture - a revised proposal, Presented at CoreGRID Integration Workshop 2005.
- [8] Gracjan Jankowski, Rafal Mikolajczak, Radoslaw Januszewski, Norbert Meyer and Maciej Stroinski, Resources virtualization in fault-tolerance and migration issues, Computational Science - ICCS 2004, 4th International Conference, Cracow, Poland, June 6-9, 2004, Proceedings, Part I.
- [9] <http://www.wcss.wroc.pl/pb/sgigrid/en/index.php>
- [10] <http://checkpointing.psnc.pl>
- [11] Rafal Mikolajczak, Radoslaw Januszewski and Gracjan Jankowski, Checkpoint Restart Packages Originated in PSNC, Presentation on TNC 2005 (published on TNC's web page).
- [12] Paul Stodghill, et. al. Use-Cases for Grid Checkpoint and Recovery, draft GGF dokument, <http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-use-cases.pdf>
- [13] Nathan Stone, Derek Simmel, Thilo Kielmann, An Architecture for Grid Checkpoint Recovery Services and a GridCPR API, draft GGF dokument, <http://gridcpr.psc.edu/GGF/docs/draft-ggf-gridcpr-architecture3.pdf>
- [14] PVM: A Framework for Parallel Distributed Computing, V. S. Sunderam, Concurrency: Practice and Experience, 2, 4, pp 315-339, December, 1990.
- [15] <http://www.csm.ornl.gov/pvm/pvm.home.html>
- [16] [http://oss.sgi.com/projects/sgi\\_propack/](http://oss.sgi.com/projects/sgi_propack/)
- [17] David Mosberger, Stephane Eranian, IA-64 Linux Kernel design and implementation, Prentice Hall PTR, 2002, ISBN 0-13-061014-3.
- [18] <http://www.sgi.com/products/servers/altix/>
- [19] <http://www.kernel.org/>
- [20] <http://lxr.linux.no/>
- [21] <http://www.cs.wisc.edu/zandy/ckpt/>