Scheduling of Computing Services on Intranet Networks

Blaise Omer Yenke, *Member*, *IEEE Computer Society*,
Jean-François Mehaut, *Member*, *IEEE Computer Society*, and
Maurice Tchuente, *Member*, *IEEE Computer Society*

Abstract—Nowadays, enterprises can provide computing services through their intranet networks by letting their available resources be used as virtual clusters for scientific computation during idle periods such as nights, weekends, and holidays. Generally, these idle periods do not permit to carry out the computations completely. It is therefore necessary to save the context of uncompleted applications for possible restart. This checkpointing mechanism is subject to resource constraints: the network bandwidth, the disk bandwidth, and the delay T imposed for releasing the workstations. We first introduce a function bw that gives the bandwidth $bw(m, V)$ of a system during the checkpointing of m applications with aggregated memory requirement V. Assuming that this bandwidth is shared equitably among the applications, the scheduling problem becomes a sequence of knapsack problems with nonlinear constraints for which we propose approximate solutions. Experiments carried out on Grid5000 show that the running time of this algorithm is negligible compared to the delay T which is of the order of few minutes. This means that the proposed scheduling algorithm does not induce a significant overhead on the checkpointing process. As a consequence, our mechanism can be incorporated in a batch scheduler.

Index Terms—Computing services, intranet networks, checkpoint scheduling, virtual clusters.

1 Introduction

During the last two decades, enterprise intranets have grown considerably. Today they often consist of several servers and hundreds of powerful workstations. The resources of this infrastructure are unused on nights, weekends, and holidays, releasing a great computing power. It would thus be judicious to exploit these long idle periods of the workstations in order to deliver computing services.

However, this infrastructure is becoming very heterogeneous, i.e., it includes various hardware, operating systems, and applications. Therefore, performing intensive computing services efficiently on such systems requires additional efforts. Two main approaches dominate the use of free resources of enterprise intranets as computation infrastructure. The first approach consists in substituting the screensaver program with an agent running the computing service. The main drawback is that the computing service may be intrusive on the data files of interactive users. In the second approach, a new operating system is launched under the control of a virtual machine. The execution of the computing service is then encapsulated in the context of a new operating system. The drawback of this approach is the great consumption of memory (two operating systems are running on the same machine, simultaneously) and the performance of computing service may be highly altered by virtualization. In this work, we adopt a new approach where machines are rebooted and integrated in a virtual computing cluster.

In our context, a virtual cluster is a dynamic infrastructure made up of distributed resources on the intranet network. An extension of such a virtual cluster can even be considered in grid environments [14]. The workstations (under Windows or Linux) retained for the virtual cluster are restarted with a diskless Linux boot with Compute-Mode [1], as described in IGGI [12]. This approach differs from the idea developed in SETI@home [2], where owners voluntarily share the CPU cycles, thanks to the fact that they incur no significant inconvenience from letting a guest process run on their machines.

On a virtual cluster, the volatility of resources is one of the constraints to be taken into account. Indeed, generally, when the available resources of the intranet of an enterprise are used as a virtual cluster for scientific computations, the idle periods do not permit to completely carry out the computations allocated to them. When some workstations currently running some applications $\{A_1, A_2, \ldots, A_n\}$ must be released within a delay T, it is necessary to save the context of these applications for possible restart. However, because of resource constraints, i.e., network and disk bandwidths, it is not always possible to checkpoint all the tasks within the delay T imposed for releasing the workstations. A good approach may be to save a collection of tasks that maximize resource consumption. We assume here that the resource consumption corresponding to a task is proportional to the computation time not yet saved. We are

- *B.O. Yenke is with the Department of Computer Science, Ngaoundere Institute of Technology, PO Box 455, Ngaoundere, Cameroon, and also with INRIA Mescal, CNRS LIG Lab., Grenoble, France. E-mail: Blaise-Omer.Yenke@imag.fr.*
- *J-F. Mehaut is with INRIA Mescal, CNRS LIG Lab., Grenoble, France. E-mail: Jean-Francois.Mehaut@imag.fr.*
- *M. Tchuente is with IRD UMI 209 UMMISCO, 32 Avenue Henri Varagnat, 93 143 Bondy Cedex, France, and also with the Faculty of Science, University of Yaounde I, UMMISCO PO Box 812, Yaounde, Cameroon. E-mail: Maurice.Tchuente@ens-lyon.fr.*

Manuscript received 15 May 2009; revised 14 Aug. 2009; accepted 23 Oct. 2009; published online 13 June 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2009-05-0133. Digital Object Identifier no. 10.1109/TSC.2011.28.

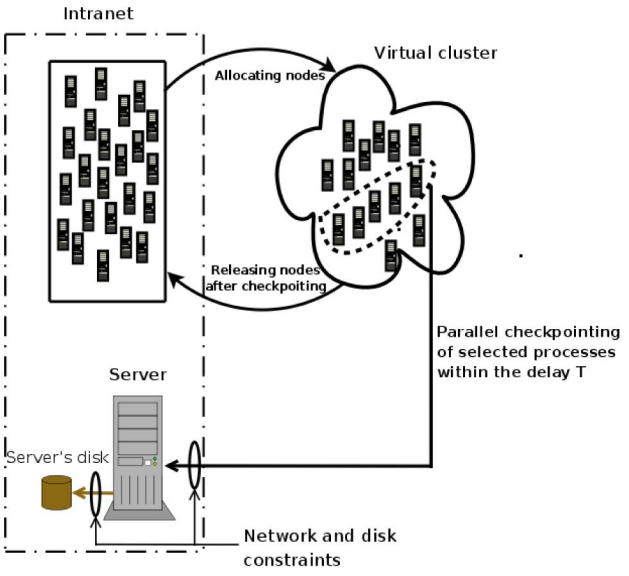

Fig. 1. Architecture of the sytsem: parallel checkpointing with constraints in virtual cluster.

thus faced with an optimization problem which consists in scheduling within the delay T, a collection of tasks $\{A_{i_1}, A_{i_2}, \ldots, A_{i_k}\}$ that maximize the sum $p_{i_1} + p_{i_2} + \cdots + p_{i_k}$ of computation time not yet saved subject to network and disk bandwidths.

It is important to note that the checkpointing time also depends on the memory requirements $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ of the checkpointed tasks. Indeed, the total required memory of an application is the most prominent component of the checkpoint image size. It is wise to make a prediction on the checkpoint time, assuming that all memory pages of an application must be saved, even if in most situations, this is not true. This prediction can be improved if we know in advance the exact amount of data to be saved. Unfortunately, such a prediction requires a great intrusion into the running processes, and can severely impact applications performances, thus it is not used here. Hereafter, we assume that it is possible to statically estimate this memory requirement.

Out-of-core algorithms refer to applications where data do not fit into the computer's memory. These algorithms [13] are explicitly optimized to fetch and access data stored in the hard disk without the swapping features of operating systems. The swapping feature is directly integrated in the algorithm. The main problem for the *out-of-core* applications is that the disk bandwidth is shared between applications' operations and checkpointing. As far as there is no local disk on the virtual cluster, the server and its disk would be a bottleneck with respect to checkpointing operations and user input/output operations. The architecture of virtual cluster is clearly unsuitable for *out-of-core* applications. Here, we assume that the application data fit into computers' memories completely, hence applications don't swap. This corresponds to *in-core* that is the most common in high performance computing, since a lot of computing clusters are dataless and even diskless.

In this study, we do not want to save locally in order to avoid disk problems (lack of space, failure, etc.), and because a workstation may not be available for a long period of time. Moreover, the checkpoint is performed on a unique server's disk as shown in Fig. 1.

The work presented here is based on the BLCR Checkpoint/Restart implementation [11] which is an Open Source, system-level checkpointer implemented as a GPL-licensed loadable kernel module for Linux.

The approach proposed here to solve the deadline constrained scheduling problem first determines the function bw that gives the aggregated bandwidth $bw(m, V)$ suitable for the parallel checkpointing of m applications of aggregated size V. Then we introduce a loop where at any time the tasks selected for checkpointing maximize $bw(m, V)$ while taking the deadline T into account.

Experiments carried out on Grid5000 [4] show that the running time of the scheduling algorithm is of the order of millisecond and is thus negligible compared to the delay T which is of the order of few minutes. This means that the proposed scheduling algorithm does not have a significant overhead on checkpointing mechanisms.

The rest of this paper is organized as follows: Section 2 presents related work. In Section 3, we give a formalization of the scheduling problem together with and approximation scheme based on a new $0/1$ knapsack problem where the constraint involves the function bw. Section 4 presents the knapsack algorithm proposed by Sartaj Sahni which combines a semienumerative approach with a greedy strategy. Section 5 gives the details of our scheduling algorithm. Some experimental results are shown in Section 6. Concluding remarks are made in Section 7.

2 Related Work

In this section, we present some known results on virtual computing environments and checkpointing strategies.

2.1 Virtual Computing Environments

The execution of intensive computation applications requiring processor resources of intranet networks can be achieved in different ways.

One approach allows end-users to explicitly donate unused resources from their workstations to a shared pool. Idle resources such as CPU cycles, memory and local storage are harvested to serve a common distributed system. When a machine is idle, an agent is started in the underlying operating system to run scientific computing. The constraint in this approach is that applications must be executed with the libraries available in the native operating system. This induces a large quantity of local storage and may interfere with the principal user of the machine. This approach is used in projects such as SETI@home [2] specialized in the search for extraterrestrial intelligence, and Folding@home [22] designed to perform computationally intensive simulations of protein folding and other molecular dynamics.

Another approach is the use of virtual machines. Virtualization technology allows to run multiple operating systems (and applications) on a single physical machine. The guest operating systems share the resources of the host machine. With this approach, there is less interference on local storage with the principal user of the machine. But running scientific computation on virtual machines can severely impact the applications' performance, since applications as well as

virtual and native operating systems all share the same RAM. VmWare [6], VirtualBox [5], and Xen [7] are open source software that provide an abstraction layer, allowing each physical machine to run several virtual machines.

The previous two approaches are interesting for environments where workstations are idle for short periods. When workstations are idle for long periods (nights, weekends, holidays), it is more advantageous to reboot the machines in order to create a virtual computing environment (virtual cluster), which is homogeneous with respect to the operating system and where resources exploited are mainly CPU cycles and RAM. This last approach avoids interference with the workstation owners. For these reasons, this approach has been adopted in this work.

2.2 Checkpointing Strategies

Checkpointing is a widely studied technique for the management of resource volatility in cluster and grid environments.

Checkpointing is clearly a mechanism which, by saving the application's context onto stable storage that is connected to the computation node through a network, prevents a process from restarting from the initial state when a system failure occurs. When restarted, a service resumes its execution from the most recent checkpoint. Many implementations (application-level, library-level, kernel-level) of checkpointing mechanisms have been proposed [11], [18], [20], [21], [25], [29], [32].

In [10], several strategies for the distributed storage of checkpoints are presented. Information dispersal and parity algorithms introduced in [26] and [27] consist in splitting data into redundant fragments. In [10], checkpoint data are stored in a single cluster. In [28], the application of predictive knowledge of resource availability to select reliable checkpoint repositories on nondedicated networks was presented. In all these studies, checkpoint data are first saved on the local disk of the computation host, before being transferred to the storage host. This approach cannot be used in our case since we are working on cluster environments where the unavailability of a workstation implies that the local resources (in particular the disk) are no longer usable. Furthermore, checkpointing followed by remote transfer induces a checkpointing overhead. To avoid this, applications are checkpointed using an end-to-end bandwidth (from the local memory of the host to the server's disk).

Batch schedulers such as Condor [33] and Sun Grid Engine [15], perform checkpointing at user-defined intervals and store checkpoints onto a set of dedicated servers.

In [9] and [35], a random distribution of failure arrivals is assumed and the ideal interval between checkpoints is computed analytically. This interval is then used in a periodic checkpointing mechanism.

The study presented here concerns the design of a module which can be incorporated in a batch scheduler, provided that checkpointing is realized using BLCR, and any releasing order comprises a delay that is given to the system to let it checkpoint the applications currently running on the targeted resources.

3 Problem Description and Approximation Scheme

This section describes in detail the problem of maximizing the aggregated computation time not yet saved under time constraint, network, and disk bandwidths constraints. An approximation scheme as a sequence of knapsack-based problems is also provided here.

3.1 Problem Description

Consider n independent applications running on a virtual cluster made of workstations of an intranet, one application per node. We assume that the workstations must be released before a delay T. We also assume that the checkpoints are saved on the server's disk—we avoid saving locally as a workstation may be unavailable for the virtual cluster for a long period of time. Because of the competition to network and disk access, it is not sure that all the applications can be checkpointed within the delay T.

Let us denote p_i the resource consumption, i.e., the computation time not yet saved of application i. The problem to be solved is to select a subset $\{i_1, i_2, \ldots, i_r\}$ of applications that can be checkpointed within the delay T and such that $p_{i_1} + p_{i_2} + \cdots + p_{i_r}$ is maximum. In this formulation, the p_is are large integers because we assume that applications are long-running ones.

Finally, we are faced with the following problem:

$$\begin{cases} maximize \sum_{i \in E} p_i \delta_i \\ subject\ to\ Checkpointing - time\{Task_i, i \in E\} \leq T. \end{cases} \quad (1)$$

There is no simple and global formulation for the constraint of this optimization problem. Indeed, any approach will first select a collection of tasks to start the checkpointing process. Later, when the checkpointing of these tasks is completed, the system will choose new candidate tasks for inclusion in the set of tasks currently being checkpointed. This leads to a loop. In the next section, we propose an example of such a loop that produces an approximate solution for this complex optimization problem.

3.2 Approximation Scheme

Consider the following notations:

$E = \{1, \ldots, n\}$ is the set of n applications to be checkpointed. T is the checkpointing delay.

$P = \{p_1, \ldots, p_n\}$ is the set of applications weights. In our context, p_i represents the time elapsed since the beginning of the execution of application i. p_i is thus the computation time not yet saved for application i.

$S = \{s_1, \ldots, s_n\}$, where s_i represents the memory requirement of application i.

$\delta = \{\delta_1, \ldots, \delta_n\}$, where $\delta_i = 1$ if application i is selected, and $\delta_i = 0$ otherwise.

n_δ is the number of applications selected, i.e., such that $\delta_i = 1$; $n_\delta = \sum_{i=1}^{n} \delta_i$.

It can be seen experimentally that when saving a distributed collection of m data blocks with aggregated size V on a unique server, the performance of a system decreases drastically when the couple (m, V) exceeds a certain threshold. This problem has been tackled in [34] where a large number of experiments have been conducted

to assess the performance of the checkpointing device when checkpointing several independent applications with BLCR.

This has led to an experimental estimation of the global bandwidth $bw(m, V)$ suitable for checkpointing m applications with aggregated memory requirement V. Let b denote the minimum between the network and the server's disk bandwidths. For any couple (m, V), we must have $bw(m, V) \leq b$. Hereafter, we assume that during the checkpointing process, the global bandwidth $bw(m, V)$ is shared equitably among the m tasks, i.e., every application is assigned a bandwidth $bw(m, V)/m$.

With these notations, the problem of selecting the processes to be checkpointed is formalized as:

$$\begin{cases} maximize \sum_{i \in I} p_i \delta_i \\ subject\ to\ (\max\{s_i \delta_i\}) \Big/ \left[bw\left(n_\delta, \sum_{i \in I} s_i \delta_i\right) \Big/ n_\delta \right] \leq T, \end{cases} \quad (2)$$

where

- $\max\{s_i \delta_i\}$ represents the largest memory requirement of the n_δ applications selected.
- $bw(n_\delta, \sum_{i \in I} s_i \delta_i)/n_\delta$ is the estimated bandwidth dedicated to the checkpointing of each of the n_δ processes selected.
- $(\max\{s_i \delta_i\})/[bw(n_\delta, \sum_{i \in I} s_i \delta_i)/n_\delta]$ represents the estimated time for the checkpointing of the n_δ processes.

This problem is more complex than the classical 0/1 knapsack problem where the constraint is just the sum of the sizes of the objects selected. The classical 0/1 knapsack problem is NP-Complete [16]. For (2), we propose an approach based on the algorithm (referred to as SS-Greedy) proposed by Sartaj Sahni [30], for this knapsack problem with nonlinear constraint.

Once a collection $\{i_1, i_2, \ldots, i_r\}$ is selected by solving (2), the system starts the checkpointing process. The task $\bar{i}$ that minimizes $(\{s_i \delta_i\})/[bw(n_\delta, \sum_{i \in I} s_i \delta_i)/n_\delta]$ will be checkpointed at time $\bar{t} = (s_{\bar{i}} \delta_{\bar{i}})/[bw(n_\delta, \sum_{i \in I} s_i \delta_i)/n_\delta]$ while the checkpointing of the other processes is still going on. Hereafter, we assume that $s_{i_1} \geq s_{i_2} \geq, \ldots, \geq s_{i_r}$. Therefore, $\bar{i} = i_r$. In order to use the bandwidth released by task $\bar{i}$, we must replace it with a task $i \notin \{i_1, i_2, \ldots, i_r\}$. This is done by solving another instance of the knapsack problem (2) with the memory requirements on $I' = I - \{\bar{i}\}$ defined as

$$s_i^{'} = \begin{cases} s_i - s_{\bar{i}}, & \text{if } i \in \{i_1, i_2, \ldots, i_r\}, \\ s_i, & \text{otherwise.} \end{cases}$$

However, since we don't interrupt ongoing checkpointing, the tasks $i \in \{i_1, i_2, \ldots, i_{r-1}\}$ have priority. Fig. 2 shows the diagram of the approximation scheme.

Let us now present the approximate algorithm introduced in [30] for the classical 0/1 knapsack problem.

4 Algorithms for the 0/1 Knapsack Problem

The 0/1 knapsack problem has been extensively studied in the literature during the past five decades. Some recent developments may be found in [17] and [19].

The 0/1 knapsack problem takes as input two sets of r positive integers $P = \{p_1, p_2, \ldots, p_r\}$ and $S = \{s_1, s_2, \ldots, s_r\}$,

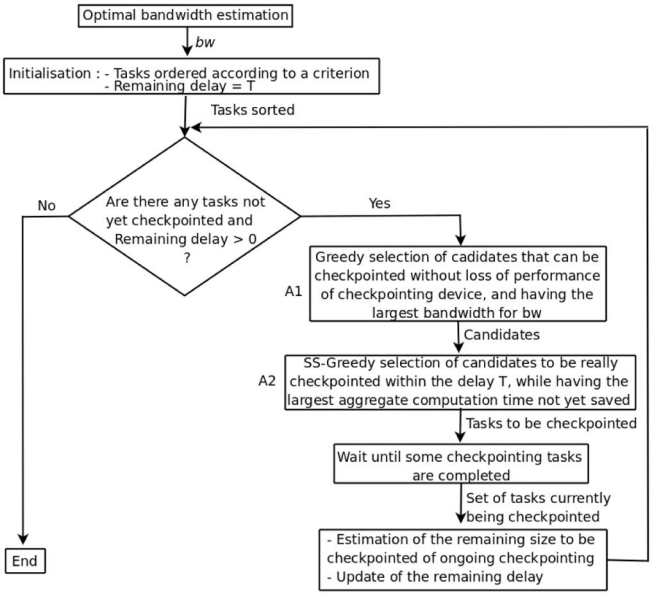

Fig. 2. Diagram of the approximation scheme.

and an integer M. The p_is can be interpreted as the profit associated with the objects $i = 1, 2, \ldots, r$, and s_i may be the size of object i. M is the size of the knapsack. If object i is put into the knapsack, the profit p_i is earned and object i occupies space s_i. The 0/1 knapsack problem is formulated as follows:

$$\begin{cases} maximize \sum_{i \in I} p_i \delta_i \\ subject\ to \sum s_i \delta_i \leq M, \end{cases} \quad (3)$$

where $\delta_i = 1$ if object i is selected, and $\delta_i = 0$ otherwise. Three main greedy strategies can be used to solve (3):

- fill the knapsack in order of decreasing densities (p_i/s_i),
- fill the knapsack in order of decreasing profits (p_i),
- fill the knapsack in order of increasing weights (s_i).

In [31], it has been shown that this problem is NP-complete, i.e., if there is a polynomial time algorithm for the knapsack problem, then one can find polynomial time solutions to a great variety of problems for which, presently, there is no known polynomial time solution.

The first polynomial-time approximation scheme for the 0/1 knapsack problem was proposed by Sartaj Sahni in [30]. In [24], Martello and Toth experimentally compared an heuristic version of their algorithm for the 0/1 knapsack problem proposed in [23] with the approximate algorithm of Sartaj. It has been shown that for small values of the number of objects, the approximate algorithm of Sartaj can produce better approximations. Since in our context, we are expecting to have a better aggregated resource consumption, and since we are working on small values of the number of objects, Sartaj approximate algorithm, though an old one, appears to be the best approach for us.

In [30], the following approximate algorithm (SS-Greedy) with parameter k_0 was given: consider all combinations $\{i_1, i_2, \ldots, i_k\}$ of k objects, $k \leq k_0$, with total size at most M. For each combination, construct a candidate by completing it using a greedy strategy. The solution produced by this algorithm is the best candidate.

5 A Scheduling Algorithm

In Section 3.2, we have proposed a formulation of the scheduling problem as a sequence of instances of the 0/1 knapsack problem (2). This sequence is implemented as a loop that is executed as long as the time delay T is not elapsed and some processes not yet checkpointed. It is important to note that an application is checkpointed uninterruptedly. However, the checkpointing of a given application can be spread over several rounds of the loop.

To achieve this goal, we proceed as follows: 1) at each iteration, determine, using a greedy strategy, a collection I of n_0 processes, $n_0 \leq n$, that can be checkpointed simultaneously without loss of performance of the checkpointing device; 2) schedule using a greedy approach, a subset of I taking into account the time constraint T.

Two criteria can be adopted to sort the set of applications: 1) in order of decreasing computation time not yet saved p_i; 2) in order of decreasing densities p_i/s_i.

In algorithm $ScheduleCkpt$ below, k_0 is the parameter introduced in Section 4. In step 1, the elapsed time t_{ckpt} since the beginning of the scheduling is initialized to zero, and the set $CKPT$ is initialized to $\emptyset$ because there is no checkpointing in progress. The set of candidates E' is initialized to E.

Algorithm $ScheduleCkpt$
1 **Init** $t_{ckpt} \leftarrow 0$; $CKPT \leftarrow \emptyset$; $E' \leftarrow E$
2 *// S' denotes the set of sizes of the processes of E'*
3 $sort(E')$
4 **repeat**
5 $\quad I \leftarrow index\_bw\_max(E')$
6 $\quad \delta \leftarrow ks\_bw(I, E', S', CKPT, T - t_{ckpt}, k_0)$
7 $\quad$ **if** $\neg is\_null(\delta)$ ***or*** $CKPT \neq \emptyset$ **then**
8 $\quad\quad CKPT \leftarrow CKPT \cup \{i \in I \,/\, \delta_i = 1\}$
9 $\quad\quad$ Start the checkpointing of all processes newly inserted into $CKPT$
10 $\quad\quad$ Let $K \leftarrow \{i \in CKPT / end\_of\_ckpt(i) = true\}$
11 $\quad\quad CKPT \leftarrow CKPT - K$
12 $\quad\quad s\_max \leftarrow max_{k \in K}\{s'_k\}$
13 $\quad\quad$ **for** ***each*** $j \in CKPT$ **do**
14 $\quad\quad\quad s'_j \leftarrow s'_j - s\_max$
15 $\quad\quad$ **end**
16 $\quad\quad E' \leftarrow E' - K$
17 $\quad\quad update(tckpt)$
18 $\quad$ **end**
19 **until** $is\_null(\delta)$ ***and*** $CKPT = \emptyset$

Let us now consider the repeat loop. The collection I of processes that can be checkpointed while preserving the performance of the system is completed in line 5 according to the greedy procedure below:

Algorithm $index\_bw\_max(E')$
1 *//The candidates $i = 1, ..., m$ of E' are supposed to be sorted* ;
2 $I' \leftarrow CKPT$; $i \leftarrow 1$;
3 $s' \leftarrow$ Aggregated size of data blocks belonging to I' ;
4 **while** $(i \leq m)$ *and* $(bw(\mid I' \mid, \sum_{j \in I'} s_j) \leq bw(\mid I' \cup \{i\} \mid, \sum_{j \in I' \cup \{i\}} s_j))$ **do**
5 $\quad I' \leftarrow I' \cup \{i\}$;
6 $\quad s' \leftarrow s' + s_i$;
7 $\quad i \leftarrow i + 1$;
8 **end**
9 return I' ;

However, among the selected processes I, only those which can be checkpointed before the deadline $T - t_{ckpt}$, will be effectively checkpointed. Procedure ks_bw does it according to the approach adopted in [30] with parameter k_0, while giving priority to the processes of $CKPT$ which are currently being checkpointed.

Line 10 is executed as soon as the checkpointing of some task $i \in K$ is ended. Lines 11 to 19 are easy to understand.

Note that procedure $index\_bw\_max$ in line 5 solves the problem stated in box A1 of Fig. 2, and procedure ks_bw in line 6 solves the problem stated in box A2.

The focal point of the complexity of algorithm $ScheduleCkpt$ is procedure ks_bw. At each loop i of the algorithm $ScheduleCkpt$, let $r_i = \mid I \mid$. In [30], it is shown that the complexity of procedure ks_bw is bounded by $O(k_0 r_i^{k_0+1})$. If algorithm $ScheduleCkpt$ runs in m loops, then the complexity of algorithm $ScheduleCkpt$ is in polynomial time bounded by $O(\sum_{i=1}^{m} k_0 r_i^{k_0+1})$.

6 Experimental Results

In this section, we describe Grid5000, the environment used to carry our experiments. Then we present the experimental curves obtained for the optimal aggregated bandwidth $bw(m, V)$, necessary to checkpoint m independent tasks of aggregated size V, and derive by interpolation a quadratic formula for $bw(m, V)$. This section ends with a detailed presentation of some checkpointing experiments.

6.1 The Environment

The experiments were conducted on Grid5000, a French infrastructure distributed on nine sites and dedicated to research in large-scale parallel and distributed systems. We have created an image containing all tools needed for checkpointing experiments (launcher, deamons for shared file system, checkpointing systems, etc.). This image was deployed on the reserved nodes, one of them acting as NFS server. The measurements were conducted on Sophia site consisting of three clusters, more precisely on Azur cluster which consists of 72 nodes, where nodes are biprocessors AMD Opteron 2 GHz, 2 GB RAM, 80 GB local disk (IDE-amd74xx). These nodes are interconnected with a Gigabit Ethernet switch and share the/home (mounted sync) of the server. With the/home synchronously mounted, we are confident that the data will immediately be written on the disk, rather than going through a buffer, which could distort the experiments.

6.2 Estimating the Aggregated Bandwidth

In [34], a large number of experiments were conducted to assess the performance of the checkpointing device with BLCR, while storing the results on a unique server's disk.

For instance, the three data sets of size 12 below were considered (the data are in MB):

$$D1 = \{105, 451, 135, 241, 329, 172, 211, 281, 117, 494, 113, 301\},$$
$$D2 = \{20, 390, 57, 129, 425, 13, 330, 19, 211, 120, 493, 312\},$$
$$D3 = \{412, 272, 320, 231, 455, 317, 401, 395, 429, 492, 201, 212\}.$$

For $i = 1, \ldots, 12$, the i first blocks were checkpointed and the experimental bandwidth bw_i registered. The curves of

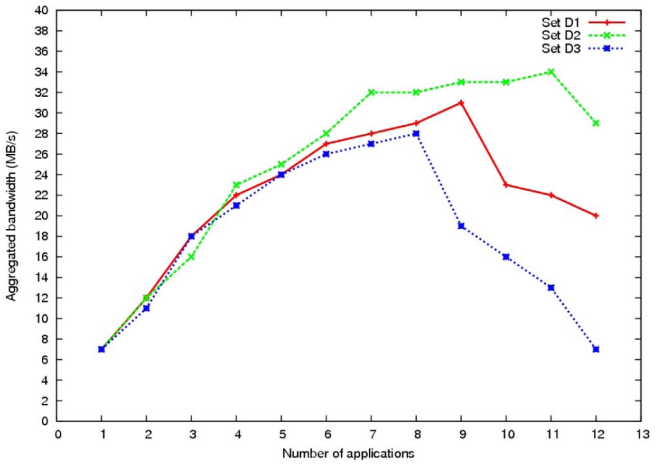

Fig. 3. Experimental aggregated bandwidths.

bw_i as a function of i are illustrated in Fig. 3. Hereafter, the peaks of these curves will be denoted $(\bar{i}, bw_{\bar{i}})$.

This inspired us to interpolate the function $(m, V) \mapsto bw(m, V)$ by a polynomial of the form

$$bw(m,V) = \sum_{0 \le i,j, \le 2} a_{ij} m^i V^j. \quad (4)$$

From the data obtained from the large number of experiments conducted in [34] with the application for the multigrid computation developed in LIG lab, we then obtained

$$bw(m,V) = am^2V^2 + bm^2 + cV^2 + dm + e,$$

with

$$a = -0.0155; b = -0.169435; c = 0.0004;$$
$$d = 5.027318; \text{ and } e = 3.753154.$$

In function $bw(m, V)$, V is expressed in GB.

With these coefficients, the surface representing bw is given in Fig. 4.

The curves of bw_i estimated as a function of i are illustrated in Fig. 5.

The experimental and the estimated peaks $(\bar{i}, bw_{\bar{i}})$ for the three examples are shown in Table 1. It can be seen that the estimations are quite accurate.

Experimental peaks can be seen in Fig. 3 and theoretical ones in Fig. 5.

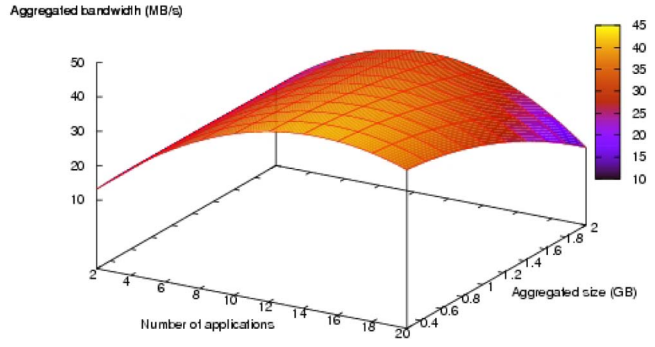

Fig. 4. Aggregated bandwidth as function of m and V.

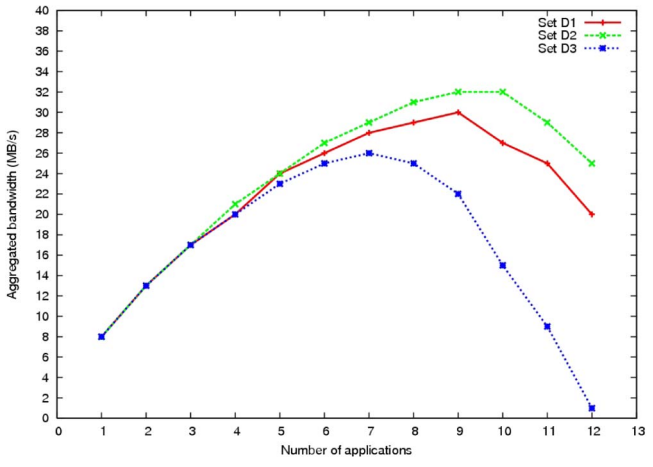

Fig. 5. Estimated aggregated bandwidths.

6.3 Scheduling Experiment

Hereafter, we denote $CTNYS$ the cumulative computation time not yet saved of applications checkpointed, $RCTNYS$ the cumulative computation time not yet saved of applications not checkpointed, nb_ckpt the number of applications checkpointed, and $sche\_time$ the time spent by the scheduling algorithm.

The validation application was written in C. The scheduling algorithm for checkpointing is implemented as a module of the multithreaded application server. We have used the launcher *taktuk* [3] to load and run the application client on all clients nodes. The application server is started on the node acting as NFS server. It takes as input, the number of processes, their sizes, and the delay T.

The server indicates to nodes clients to execute the application benchmark with the specified size. The application that each node client must execute is a synthetic benchmark whose all memory pages must be saved. Each node client receives an order and creates a child process whose context is replaced by that of the application it executes while preparing the application to be checkpointed with BLCR. This allows the application client to obtain the PID of the process to be checkpointed.

To avoid conflicts that could arise if two different clients obtained the same PID for the application they must checkpoint, we have created a directory (the hostname of the client) for each client on the/home of the server. The application to be performed by each node client is placed in that directory, which allows each client to execute and save the benchmark application in that directory.

After the launching phase, the application server invokes the scheduling checkpointing module that determines the processes to be checkpointed according to the

TABLE 1
Measured and Estimated Peaks

	D1	D2	D3
$bw_{\bar{i}}$ measured (MB/s)	31.20	34.03	28.77
$bw_{\bar{i}}$ estimated (MB/s)	30.04	32.53	26.24
$\bar{i}$ measured	9	11	8
$\bar{i}$ estimated	9	10	7

TABLE 2
Parallel Checkpointing with $ScheduleCkpt$

$n = 50 : T = 300 : P = \{26, 27, ..., 75\}$			
Size (MB)	nb ckpt	CTNYS (s)	RCTNYS (s)
10	50	2525	0
50	50	2525	0
100	50	2525	0
200	45	2385	140
400	28	1722	803

TABLE 3
Greedy Sequential Checkpointing

$n = 50 : T = 300 : P = \{26, 27, ..., 75\}$			
Size (MB)	nb ckpt	CTNYS (s)	RCTNYS (s)
10	50	2525	0
50	50	2525	0
100	16	1080	1445
200	8	572	1953
400	4	297	2228

TABLE 4
Results of $ScheduleCkpt$ with Criterion p_i

$ScheduleCkpt$	$n = 25 : T = 180(s)$		
	CTNYS (s)	RCTNYS (s)	sche_time (μs)
$k_0 = 0$	4842	5362	28
$k_0 = 1$	6218	3996	1834
$k_0 = 2$	6833	3371	20776

TABLE 5
Results of $ScheduleCkpt$ with Criterion p_i/s_i

$ScheduleCkpt$	$n = 25 : T = 180(s)$		
	CTNYS (s)	RCTNYS (s)	sche_time (μs)
$k_0 = 0$	5021	5183	31
$k_0 = 1$	6947	6947	2565
$k_0 = 2$	7135	7135	21090

algorithm described in this work. The clients receive an order to checkpoint application running on the node. At the end of the checkpoint, they inform the server and await any other order. When it is no longer possible to checkpoint any application and when there is no more ongoing checkpointing, the server informs nodes clients to kill the process they perform.

In all implementations, the measurements made during the scheduling were stored in trace files for later use.

In this section, two criteria are considered: selecting the applications in order of decreasing computation time not yet saved p_i; selecting the applications in order of decreasing densities p_i/s_i representing the ratio between computation time not yet saved and memory requirement.

In Tables 2 and 3 below, applications have identical sizes. Experiments were conducted with $n = 50$ and $T = 300$.

It can be seen from Tables 2 and 3 that parallel checkpointing is better than sequential checkpointing.

In our experiments, for $k_0 \geq 3$, the time for the scheduling was of the same order of the delay T imposed for releasing the resources, which is not reasonable. Consequently, in the results that follow, $k_0 < 3$.

In the experiments that follow, a large number of tests were conducted on different data sets, in three families of size $n = 25$, $n = 50$, and $n = 70$. In each family, the results

TABLE 6
Results of $ScheduleCkpt$ with Criterion p_i

$ScheduleCkpt$	$n = 50 : T = 300(s)$		
	CTNYS (s)	RCTNYS (s)	sche_time (μs)
$k_0 = 0$	17340	12207	87
$k_0 = 1$	21633	7914	6350
$k_0 = 2$	24260	5287	72542

TABLE 7
Results of $ScheduleCkpt$ with Criterion p_i/s_i

$ScheduleCkpt$	$n = 50 : T = 300(s)$		
	CTNYS (s)	RCTNYS (s)	sche_time (μs)
$k_0 = 0$	19200	10347	145
$k_0 = 1$	22985	6562	5478
$k_0 = 2$	27103	2444	65983

TABLE 8
Results of $ScheduleCkpt$ with Criterion p_i

$ScheduleCkpt$	$n = 70 : T = 420(s)$		
	CTNYS (s)	RCTNYS (s)	sche_time (μs)
$k_0 = 0$	15895	26901	169
$k_0 = 1$	24454	18342	52503
$k_0 = 2$	26900	15896	269430

TABLE 9
Results of $ScheduleCkpt$ with Criterion p_i/s_i

$ScheduleCkpt$	$n = 70 : T = 420(s)$		
	CTNYS (s)	RCTNYS (s)	sche_time (μs)
$k_0 = 0$	18952	23844	232
$k_0 = 1$	25677	17119	39862
$k_0 = 2$	28734	14062	292110

were almost similar. Since the results cannot all be presented, we exhibit one in each family that illustrates the performance of the scheduling algorithm. For the three families, the memory requirements s_i of applications were randomly generated between 100 and 500 MB.

Table 4 (respectively, Table 5) gives the performance of $ScheduleCkpt$ with criterion p_i (respectively, criterion p_i/s_i). $n = 25$, $T = 180$ s (3 minutes) and the computation time not yet saved p_i were generated between 60 s (1 minute) and 600 s (10 minutes).

Table 6 (respectively, Table 7) gives the performance of $ScheduleCkpt$ with criterion p_i (respectively, criterion p_i/s_i). $n = 50$, $T = 300$s (5 minutes) and p_i were generated between 300 s (5 minutes) and 1,800 s (30 minutes).

Table 8 (respectively, Table 9) gives the performance of $ScheduleCkpt$ with criterion p_i (respectively, criterion p_i/s_i). $n = 70$, $T = 420$ s (7 minutes) and p_i were generated between 300 s (5 minutes) and 1,800 s (30 minutes).

Tables 4, 5, 6, 7, 8 and 9 show that the quality of the scheduling algorithm increases with k_0. We can also note that the scheduling time increases with k_0 but remains negligible when compared to the delay T.

On the other hand, it is better to sort the application with respects to the p_i/s_i.

7 Conclusion and Future Work

In this paper, we have considered a context where the available resources of the intranet of an enterprise are used as

a virtual cluster for scientific computation during idle periods such as nights, weekends, and holidays. Generally, these idle periods do not permit to completely carry out the computations. It is therefore necessary to save the context of uncompleted applications for possible restart, but it is not always possible to save all of them because of time constraint. Assuming that these applications are independent, we are faced with the problem of checkpointing a subset among n applications running on workstations that must be released before a delay T, while maximizing the aggregated computation time not yet saved. We have first proposed for Grid5000, which was our experimental environment, a function bw that gives the bandwidth $bw(m, V)$ of the system during the parallel checkpointing of m applications with aggregated memory requirement V. This function was used to estimate the checkpointing time, assuming that the bandwidth $bw(m, V)$ is shared equitably among the m tasks. For the deadline-constrained scheduling problem, we then proposed an approximation scheme consisting of a loop where in every pass, we first select m candidate tasks in order to optimize the network bandwidth $bw(m, V)$, and we then choose for checkpointing within the delay T, the candidates that maximize the aggregated computation time not yet saved.

Finally, we proposed a mechanism where the selection of candidates follows a greedy strategy. The choice of tasks to be checkpointed is done according to the Sartaj Sahni approach that generates all combinations of k_0 candidates ($k_0 < 3$), completes these k_0 candidates following a greedy approach while fulfilling the deadline constraint T, and chooses the best solution, i.e., the one that maximizes the aggregated computation time not yet saved. It is important to note that we did not propose the Sartaj Sahni algorithm for the selection of candidates because, since the number of tasks may be large, the execution time of Sartaj Sahni algorithm may be prohibitive.

Experiments carried on Grid5000 show that our algorithm performs better that the sequential approach (the aggregated computation time not yet saved is doubled), and much better than the totally parallel checkpointing which induces a drastic decrease of the system bandwidth. Experiments also show that the running time of the scheduling algorithm, which is of the order of millisecond, is negligible compared to the delay T which is of order of few minutes. This means that the proposed scheduling algorithm does not have a significant overhead on checkpointing mechanisms.

This paper raises four interesting questions. The first question concerns a theoretical explanation of the polynomial formula established experimentally for bw on Grid5000. The second is to compute bw for other networks and then to see whether there is a general polynomial formula for these functions. The third question deals with the design of efficient algorithms for the new types of knapsack problems presented here and where the constraints are nonlinear. The last question concerns systems where the checkpoints are saved on several servers. Indeed, checkpointing on a unique server, as assumed here, may incur performance bottleneck and a single point of failure. The problem of checkpointing on several servers is therefore an important question. Following the approach introduced here, this problem may need, for instance in the particular case of two servers, the construction of function $(m_1, V_1, m_2, V_2) \longmapsto bw(m_1, V_1, m_2, V_2)$ that gives the bandwidth of the system when $m_1 + m_2$ tasks of aggregated memory requirement $V_1 + V_2$ are saved in parallel, m_1 on the first server and m_2 on the other one.

From a more practical point of view, the next step of this research will be the integration of our scheduling module in a batch scheduler such as OAR [8], provided that checkpointing is realized using BLCR, and any releasing order comprises a delay that is given to the system to let it checkpoint the applications currently running on the targeted resources.

Acknowledgments

The authors would like to heartily thank the anonymous reviewers for their insightful comments which have led to improvements on this work.

References

[1] http://computemode.imag.fr, 2011.
[2] seti@home: Search for Extraterrestrial Intelligence at Home, http://setiathome.ssl.berkeley.edu, 2011.
[3] http://taktuk.gforge.inria.fr, 2011.
[4] http://www-sop.inria.fr/aci/grid/public/library/rapport-grid5000-v3.pdf, 2011.
[5] http://www.virtualbox.org, 2011.
[6] http://www.vmware.com, 2011.
[7] http://www.xen.org, 2011.
[8] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard, "A Batch Scheduler with High Level Components," *Proc. Fifth Int'l Symp. Cluster Computing and Grid (CCGrid '05)*, May 2005.
[9] J.T. Daly, "A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303-312, 2006.
[10] R.Y. de Camargo, R. Cerqueira, and F. Kon, "Strategies for Checkpoint Storage on Opportunistic Grids," *IEEE Distributed Systems Online*, vol. 7, no. 9, p. 1, Sept. 2006.
[11] J. Duell, P. Hargrove, and E. Roman, "The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart," Technical Report LBNL-54941, Berkeley Lab, Nov. 2003.
[12] F. Dupros, F. Boulahya, J. Vairon, P. Lombard, N. Capit, and J-F. Mehaut, "IGGI, a Computing Framework for Large Scale Parametric Simulations: Application to Uncertainty Analysis with Toughreact," *Proc. Tough Symp.*, 2006.
[13] C. Eddy and U. Gil, "On the Performance of Parallel Factorization of Out-of-Core Matrices," *Parallel Computing*, vol. 30, no. 3, pp. 357-375, Feb. 2004.
[14] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual Clusters for Grid Communities," *Proc. Sixth IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '06)*, pp. 513-520, 2006.
[15] W. Gentzsh, "Sun Grid Engine: Towards Creating a Compute Power Grid," *Proc. Int'l Symp. Cluster Computing and Grid (CCGrid '01)*, pp. 35-39, 2001.
[16] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Computer Science, 1978.
[17] A.J. Bryant, "Greedy, Genetic, and Greedy Genetic Algorithms for the Quadratic Knapsack Problem," *Proc. Conf. Genetic and Evolutionary Computation (GECCO '05)*, pp. 607-614, June 2005.
[18] J. Janakiraman, J.R. Santos, D. Subhraveti, and Y. Turner, "Cruz: Application-Transparent Distributed Checkpoint-Restart on Standard Operating Systems," *Proc. Int'l Conf. Dependable Systems and Network (DSN '05)*, June 2005.
[19] R. Kumar, A.H. Joshi, K.K. Banka, and P.I. Rockett, "Evolution of Hyperheuristics for the Biobjective 0/1 Knapsack Problem by Multiobjective Genetic Programming," *Proc. 10th Ann. Conf. Genetic and Evolutionary Computation (GECCO '08)*, July 2008.
[20] O. Laadan and J. Nieh, "Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems," *Proc. USENIX Ann. Technical Conf.*, pp. 323-336, June 2007.

[21] O. Laadan, D. Phung, and J. Nieh, "Transparent Checkpoint-Restart of Distributed Apications on Commodity Clusters," *Proc. IEEE Int'l Conf. Cluster Computing,* Sept. 2005.
[22] S.M. Larson, C.D. Snow, M.R. Shirts, and V.S. Pande, "Folding@Home and Genome@Home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology," http://arxiv.org/abs/0901.0866v1, 2003.
[23] S. Martello and P. Toth, "A New Algorithm for the 0-1 Knapsack Problem," *J. Management Science,* vol. 34, no. 5, pp. 633-644, 1988.
[24] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley and Sons, 1990.
[25] J.S. Plank, "An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance," Technical Report UT-CS-97-372, Dept. of Computer Science, Univ. of Tennessee, July 1997.
[26] J.S. Plank, K. Li, and M.A. Puening, "Diskless Checkpointing," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 10, pp. 972-986, Oct. 1998.
[27] M.O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Taulerence," *J. ACM,* vol. 36, no. 2, pp. 335-348, 1989.
[28] X. Ren, R. Eigenmann, and S. Bagchi, "Failure-Aware Checkpointing in Fine-Grained Cycle Sharing Systems," *Proc. 16th Int'l Symp. High performance Distributed Computing (HPDC '07),* June 2007.
[29] E. Roman, "A Survey of Checkpoint/Restart Implementation," technical report, Publication LBNL-54942C, Berkeley Lab, 2002.
[30] S. Sahni, "Approximate Algorithms for the 0/1 Knapsack Problem," *J. ACM,* vol. 22, no. 1, pp. 115-124, Jan. 1975.
[31] S. Sahni, "Some Related Problems from Network Flows, Game Theory and Integer Programming," *Proc. IEEE 13th Ann. Symp. Switching and Automata Theory,* Oct. 1972.
[32] S. Sankaran, J.M. Squyres, B. Barrett, and A. Lumsdaine, "The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing," *Proc. LACSI Symp.,* Oct. 2003.
[33] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency—Practice and Experience,* vol. 17, nos. 2-4, pp. 323-356, 2004.
[34] B.O. Yenke, "Prédiction Des Performances Des Opérations de Sauvegarde/Reprise Sur Cluster Virtuel," *RENPAR '18/SympAAA 2008/CFSE '3/Fribourg, Suisse,* du 11 au 13 février, 2008.
[35] A. Ziv and J. Bruck, "An On-Line Algorithm for Checkpoint Placement," *IEEE Trans. Computer,* vol. 46, no. 9, pp. 976-985, Sept. 1997.