Optimum Checkpoints for Time and Energy

Erol Gelenbe[1], Miltiadis Siavvas[2], Pawel Boryszko[1], Joanna Domanska[1]
[1)] IITIS, Polish Academy of Sciences
ul. Baltycka 5, 44100 Gliwice, Poland
[2)] Information Technologies Institute, Centre for Research and Technology Hellas
6th km Harilaou-Thermi, Thessaloniki, 57001 Greece

***Abstract*—We consider programs running on systems where failures can occur, and in which checkpoints are used to assure the reliability of the programs' execution. We consider the energy consumption for the program's execution, in addition to its run time, as criteria that need to be used to minimize the overall cost due to checkpoints. New expressions for both the program's run-time and the corresponding energy consumption that include the failure probability per instruction execution, and the overhead incurred for each checkpoint, are derived. For programs which run indefinitely in an infinite loop, such as certain real-time applications, a first principle based analysis yields the checkpoint interval that minimizes a linear combination of the average execution time of the program and of its average energy consumption via the classical "Lambert W(.) function". For looping programs which run for a finite time, the optimum checkpoint interval is obtained using a different approach. The results are illustrated with numerical examples.**

I. Introduction

Reliability is a critical aspect for long-running software applications that perform computationally expensive tasks [1]. In such applications, a single failure may lead to the re-execution of a large number of operations, leading to significant overheads in important Quality-of-Service (QoS) parameters, including performance and energy consumption.

These overheads may have significant consequences for time-critical software programs that have to provide their results within a specific time frame [2], [3], [4], as well as for energy-critical applications that have to ensure the energy efficiency of their operation [5]. Hence, fault tolerance mechanisms that avoid complete program re-execution in case of failures are required.

Several such fault tolerance mechanisms exist [6], and the Checkpoint and Restart (CR) is the most widely used. CR periodically keeps a safe copy (or a checkpoint) of the program execution state, and uses it to restart the program in case of a failure.

Checkpointing schemes have also become very popular in high performance computing systems [7], [8], [9], [10], and they have also been impelemented in operating systems such as Unix or Linux [11], [12], [13]. Another area where checkpointing has proved to be important, is the consistency of distributed systems [14], [15], [16].

This research was supported by the European Commission through the Horizon 2020 SDK4ED Project under Grant Agreement No. 780572. The contents of this paper represent the opinions of the authors, and do not engage the responsibility of the European Commission.

Among the existing checkpointing strategies [6], [17], the Application-level Checkpoint and Restart (ALCR) [18], [19] is considered the most efficient, as it leaves a smaller memory footprint. However, it implementing checkpoints requires significant programming knowhow and effort. Also, expertise is needed to select both the code locations and the checkpoint intervals. Although existing ALCR tools and libraries facilitate the insertion of checkpoints in long-running loops [20], [21], these tools typically do not provide a method to select the inter-checkpoint interval, which has a key influence on the execution time and energy consumption of the software.

Long intervals of time between checkpoints will increase the overhead associated with system restart, while short intervals will increase the overhead caused by the checkpoints themselves. Hence, the checkpoint interval must be chosen so as to optimize important QoS parameters, including performance and energy consumption [22], [23], [24].

Although the impact of the checkpoint interval has been extensively studied with regard to overall system availability and total program execution time in the case of transaction-oriented systems [25], [26], [27], [28], [29], there has been hardly any work on checkpoint optimization for energy consumptin. There has also been little work about checkpoint optimization for modern software applications that adopt the ALCR mechanism to the exception of some recent papers [30], [31].

The present paper innovates mainly by studying checkpoint intervals in order to effect savings in the total energy consumed by a program's execution, in addition to the usual consideration of total execution time. Indeed, increasingly over the last decade the question of energy consumption has come to the forefront both for Clouds and data centres [32], [33], [34], [35] and with regard to the sustainability of hardware and software systems in general [36], [37].

In the sequel, starting from first principles, we develop a mathematical model to estimate the average execution time as well as the energy consumption of a program with long loops that operates in the presence of failures, without and with application-level checkpointing. This model is used to compute the checkpoint interval that minimizes the average execution time of the program, as well as the checkpoint interval that minimizes its energy consumption.

More importantly, it is also used to determine the checkpoint interval that can achieve a compromise between expected execution time and energy consumption, by minimizing a cost function that combines these two elements. Analytical

expressions of the resulting optimum checkpoint interval are obtained in terms of the well known Lambert Function. Several numerical examples illustrate these results. We also present a toolbox that can be used to select the checkpoint interval that minimizes these various quantities.

II. A SINGLE LOOP ROGRAM WITH CHECKPOINTS

Consider a program P that executes y_n instructions betwen its $(n-1)$-th and n-th checkpoint, without counting all possible failures and failure recoveries. Now consider the instant $t_n > 0$ when the program creates its $n-th$ checkpoint, and let Y_n denote the total number of instructions that the program has executed by time t_n since it started, where Y_n does not include all the repeated instructons that were executed due to checkpoints and failure recovery, and onviously:

$$Y_n = \sum_{i=1}^{n} y_i. \tag{1}$$

Let $B(Y_n)$ be the time needed to create the $n-th$ checkpoint. This quantity will generally depend on the total memory space occupied by the program, which is fixed once the program is known, but in certain cases it may depend on Y_n, since the program may generate new data as it is executing. Hence we will write $B(Y_n) = B_0 + B_1 Y_n$ where $B_0 > 0$ and $B_1 \geq 0$ are constants for the given program.

On the other hand, suppose a failure occurs after the program has successfully executed y instructions after a checkpoint. For instance suppose a failure occurs after the program has executed $Y_n + y$ instructions. If $b(Y_n, y)$ is the time it requires to restart the program from the most recent checkpoint, when the program has successfully executed $y \leq Y_{n+1} - Y_n$ instructions after the most recent checkpoint but before the $(n+1)$ checkpoint, then we will take $b(Y_n, y) = b_0 + b_1 y$ where $b_0 > 0$ and $b_1 \geq 0$ are constants, so that this time depends only on the number of instructions that have been executed by the program since the last checkpoint was established.

In summary we are assuming that:

- The time $B^c(Y_n)$ needed to establish the n-th checkpoint depends on the "age of the program" or the total number of instructions Y_n it has executed since the beginning, i.e. $B^c(Y_n) = B_0^c + B_1^c(Y_n)$,
- The time $b^c(Y_n, y)$ needed to recover from a failure after the n-th checkpoint, including the time related to reloading system state after the failure, only depends on $y <\leq Y_{n+1} - Y_n$, the "computation time undertaken by the program since the last checkpoint", i.e. $b^c(Y_n, y) \equiv b^c(y) = b_0^c + b_1^c y$.

Similarly, we denote the energy consumption for creating the n-th checkpoint to be $B^e(Y_n)$, and $b^e(y)$ is the energy used to recover from a failure after a failure that occurs when the total number of instructions executed is $Y_n + y\ leq Y_{n+1}$. Also, we will have $B^e(Y_n) = B_0^e + B_1^e Y_n$, and $b^e(y) = b_0^e + b_1^e y$ with $B_0^e > 0,\ B_1^e \geq 0$ and $b_0^e > 0,\ b_1^e \geq 0$.

Let $\alpha, \beta > 0$ be positive constants that represent the relative costs of computation time and energy consumption. We can then define the parameters:

$$\begin{aligned} B_0(Y) &= \alpha B_0^c(Y) + \beta B_0^e(Y), \\ B_1(Y) &= \alpha B_1^c(Y) + \beta B_1^e(Y), \\ b_0 &= \alpha b_0^c + \beta b_0^e, \\ b_1 &= \alpha b_1^c + \beta b_1^e,\ c = \alpha c^c + \beta c^e. \end{aligned}$$

A. Fixed Checkpoint Intervals

Earlier work has shown that "age dependent" checkpoints [28] can reduce the overall cost of checkpointing and failure recovery, when (for instance) the failure rate of a system increases with time. However, most practical checkpointing schemes use a simpler approach where checkpoints are carried out periodically each time the program has executed successfully a predertmined fixed number of instructions $y_n = y$. Thus, in the sequel we will make this assumption, so that checkpoints are placed after $Y_1 = y,\ Y_2 = 2y,\ ..\ Y_n = ny$, etc. instructions have been successfully executed, and we will proceed to compute the optimum value of y, assuming that n is fixed in advance.

When the program ends after $Y = Ny$ instructions are executed, a further $(N+1)$-th checkpoint is not needed, while the first checkpoint is obviously installed before the first instruction is executed.

We can then formulate our problem as that of a program that executes a total fixed number of instructions Y, where we want to choose the constant value y of the number of instructions between checkpoints, or equivalently we can choose N, the number of checkpoints so that $Y = Ny$ so that the total overhead in additional work and energy consumption due to failures and due to checkpoints is minimized.

For a given y, let us compute $C^c(y)$, the corresponding *"total expected execution time"*, including all restarts due to failures, starting from the most recent checkpoint. When the average execution time per instruction is c, and the failure probability per instruction is $(1-a)$, the total average time elapsed time for the execution of y instructions is:

$$\begin{aligned} C^c(y) &= cya^y + (b_0 + C^c(y))(1 - a^y), \\ &\quad +(c^c + b_1^c)\sum_{x=1}^{y} x a^{x-1}(1-a), \end{aligned} \tag{2}$$

because with probability a^y a failure does not occur durung the y instructions, leading to an execution time of $c^c y$ time units, while with probability $(1-a)^y$ at least one failure does occur among the y instructions, and the first of those requires a program re-start time of b_0^c, to which we should add $C^c(y)$ representing the effect of all future failures after the program has been re-initialised from the checkpoint.

In addition, we have to include the execution time plus the amounf of additonal work needed per executed instruction, until the failure occurs – hence the term $(c^c + b_1^c)$ – multiplied

by x and the probability that the failure occurs at instruction x which is $a^{x-1}.a$, summed over x running from 1 to y. Since

$$\begin{aligned}\sum_{x=0}^{y} a^x &= \frac{1-a^{y+1}}{1-a},\\ and\ \frac{d}{da}\frac{1-a^{y+1}}{1-a} &= \frac{1-ya^y(1-a)-a^y}{(1-a)^2},\end{aligned}$$

we obtain:

$$C^c(y) = b_0^c[a^{-y}-1] + \frac{c^c+b_1^c}{1-a}[a^{-y}-1] - b_1^c y. \quad (3)$$

the total expected energy consumption $C^e(y)$ for a number of instructions y after the most recent checkpoint, we similarly obtain the quantity:

$$C^e(y) = b_0^e[a^{-y}-1] + \frac{c^e+b_1^e}{1-a}[a^{-y}-1] - b_1^e y, \quad (4)$$

where c^e denotes the average energy consumption per instruction, so that

$$\begin{aligned}C(y) &= \alpha C^c(y) + \beta C^e(y),\\ &= b_0[a^{-y}-1] + \frac{c+b_1}{1-a}[a^{-y}-1] - b_1 y. \quad (5)\end{aligned}$$

Interestingly enough, we can show using l'Hôpital's Rule, for all $y \geq 1$, that:

$$\lim_{a\to 1} C(y) = c^y, \quad (6)$$

as would be expected.

Treating y as if it were a real number, we can compute the derivative of $C(y)$. We first note that for a differentiable function $f(y)$ of the real variable y, we can write:

$$\begin{aligned}\frac{df}{dy} &= f.\frac{d\ln f}{dy},\ hence\\ \frac{d}{dy}a^{-y} &= -a^{-y}.\ln a, \quad (7)\end{aligned}$$

and therefore

$$\frac{dC(y)}{dy} = -\ln a.[\frac{b_0}{a^y} + \frac{c+b_1}{a^y(1-a)}] - b_1. \quad (8)$$

Because $a \leq 1$, the quantity $-\ln a \geq 0$, and since y is large, $\frac{1}{a^y}$ is very large and $\frac{dC(y)}{dy} > 0$.

III. Minimizing Computation Time and Energy

When we include both the time and energy needed to create each checkpoint, and assuming a fixed number of instructions y executed between successive checkpoints, we can obtain the total cost of the program up to and including the last instruction executed at $Y = yN$ as:

$$\begin{aligned}K_N(y) &= NB_0 + \sum_{i=1}^{N} iyB_1 + NC(y), \quad (9)\\ &= NB_0 + C(y) + \frac{N(N+1)}{2}yB_1. \quad (10)\end{aligned}$$

The optimum checkpoint interval y^* is then the value of y that minimises $\kappa_N(y)$, the overall cost per unit work that is accomplished, i.e. $K_N(y)$ divided by $Y = Ny$ which is the total number of useful instructions executed over this time:

$$\begin{aligned}\kappa_N(y) &\equiv \frac{K_N(y)}{Ny}\\ &= \frac{B_0+C(y)}{y} + (\frac{Y}{y}+1)\frac{B_1}{2}\\ &= \frac{B_0+\frac{B_1Y}{2}+C(y)}{y} + \frac{B_1}{2}. \quad (11)\end{aligned}$$

Therefore to seek the optimum value of y, we compute the following derivative and set it to zero:

$$\frac{d\kappa_N}{dy} = \frac{y\frac{dC(y)}{dy} - (B_0 + \frac{B_1Y}{2} + C(y))}{y^2}, \quad (12)$$

so that the optimum value of y is:

$$\begin{aligned}y^* &= \frac{B_0+\frac{B_1Y}{2}+C(y^*)}{\frac{dC(y)}{dy}|_{y=y^*}} =\\ &\frac{B_0+\frac{B_1Y}{2}+b_0[a^{-y^*}-1]+\frac{c+b_1}{1-a}[a^{-y^*}-1]-b_1y^*}{-\ln a.[\frac{b_0}{a^{y^*}}+\frac{c+b_1}{a^{y^*}(1-a)}]-b_1},\\ or\ &-\frac{y^*lna+1}{a^{y^*}} = \frac{B_0+\frac{B_1Y}{2}}{b_0+\frac{c+b_1}{1-a}} - 1. \quad (13)\end{aligned}$$

Defining $B = B_0 + \frac{B_1Y}{2}$ and

$$A = b_0 + \frac{c+b_1}{1-a}, \quad (14)$$

we have:

$$\begin{aligned}-\frac{y^*\ln a+1}{a^{y^*}} &= \frac{B_0+\frac{B_1Y}{2}}{b_0+\frac{c+b_1}{1-a}} - 1,\\ or\ \ln(a^{-y^*}.e^{-1})[e^{-1}a^{-y^*}] &=\\ -(y^*\ln a+1)e^{-(y^*\ln a+1)} &= \frac{B-A}{e.A}, \quad (15)\end{aligned}$$

To verify that y^* is the minimum value, we compute:

$$\frac{d^2\kappa_N(y)}{dy^2} = \frac{y^3C''(y) - 2y(yC'(y) - B - C(y))}{y^4}, \quad (16)$$

where $C'(y)$, $C''(y)$ denote the first and second derivatives of $C(y)$ with respect to y, and $B = B_0 + B_1Y$. Since at y^* we have $y^*C'(y^*) = B + C(y^*)$, we can write:

$$\frac{d^2\kappa_N(y)}{dy^2}|_{y=y^*} = \frac{C''(y)}{y}|_{y=y^*}, \quad (17)$$

and we need to examine the sign of $C''(y^*)$. Starting from (8) we have:

$$C''(y) = a^{-y}(\ln a)^2[b_0 + \frac{c+b_1}{1-a}] - a^{-y}\ln a\frac{c+b_1}{(1-a)^2}, \quad (18)$$

which is positive, so that y^* is indeed the value of y at the minimum.

A. The Optimum Checkpoint using the Lambert Function

Let us first recall the definition of the *Lambert Function* $W(z)$ [38], [39], [40], [41]. Consider any two numbers $z,\ w$, which have the following relation:

$$z = w \exp w; \Longleftrightarrow w = W(z). \tag{19}$$

Thus, if we can write $z = we^w$, then $w = W(z)$, and similarly if $w = W(z)$, then $z = we^w$.

Applying (19) to equation (15), we can write the expression for y^* as:

$$y^* = -\frac{1}{\ln a}[W(\frac{B-A}{e.A}) + 1], \tag{20}$$

which provides an explicit solution for the value of the optimum checkpoint interval y^*. Cleaarly, if we set $\alpha = 1$ and $\beta = 0$, we obtain the optimum checkpoint that simply minimizes the overall exection time, without consideration for the energy consumption.

Also, if in the system under consideration the creation of a checkpoint does not depend on the amount of successful computation that the program has accomplished until the time of the checkpoint, then we simply set $B_1^c = B_1^e = 0$ in the expression for B, so that $B = B_0$ which is the case that is usually discussed in the literature.

B. Sensitivity of the Optimum to Energy Consumption and Computation Time

An important question concerns how y^* varies with changes in the relative importance of the energy expenditure with respct to computation time. To address this issue as a single parameter problem, we will set $\alpha = 1$, and consider the derivative of y^* with respect to β. Noting that we can now write $B = B^c + \beta B^e$ and $A = A^c + \beta A^e$, we have:

$$\begin{aligned}
\frac{\partial y^*}{\partial \beta} &= -\frac{1}{\ln a} W'(\frac{B-A}{e.A}).\frac{B^eA^c - A^eB^c}{(eA)^2}, \\
&= -\frac{1}{\ln a}\frac{W(\frac{B-A}{e.A})(B^eA^c - A^eB^c)}{\frac{B-A}{e.A}(1+W(\frac{B-A}{e.A}))((eA)^2)}, \\
&= -\frac{(y^* \ln a + 1)(B^eA^c - A^eB^c)}{y^*(\ln a)^2(B-A)eA},
\end{aligned} \tag{21}$$

where we have used the identity:

$$\frac{dW(x)}{dx} = \frac{W(x)}{x(1+W(x))}, \tag{22}$$

when $x \neq 0$ and $x \neq -\frac{1}{e}$. These two conditions will be satisfied because it is unlikely in practice that the system parameters be such that $B = A$, furthermore it is impossible that $B - A = -A$ because $B > 0$.

Thus we can use the expression (21) to determine how fast y^* will vary as a function of y^*. In particular we have the following very interesting result.

Result: When $B^eA^c = A^eB^c$, then y^* is does not depend on the relative weight of the execution time and energy consumption, so that a single value of y^* will minimize the overall cost for $\alpha = 1$ and any value of β that represents the relative importance of energy consumption to computtaion time.

IV. A Program with a Single Long Loop

In this section we will apply the previous results to a program with a single long loop of length L instructions which is executed some number, say T times, so that $Y = LT$. For this program we may be constrained to place checkpoints either at the start of a loop so that $y = m.L$ with one checkpoint for each $m > 1$ loops, or n checkpoints may be placed within the loop with $L = ny$ where $n > 1$, or we set $n = 1$.

We first apply the previous results to compute y^*:

$$y^* = -\frac{1}{\ln a}[W(\frac{B-A}{e.A}) + 1], \tag{23}$$

where:

$$B = B_0 + B_1 L.T,\ A = b_0 + \frac{c + b_1}{1-a}, \tag{24}$$

so that

$$y^* = -\frac{1}{\ln a}[W\big(\frac{(1-a)(B_0 + B_1 L.T)}{e[b_0(1-a) + c + b_1]} - \frac{1}{e}\big) + 1]. \tag{25}$$

Let us denote by $I(x)$ the integer that is closest to the real number x. Then we compute $r = \frac{L}{y^*}$, and:

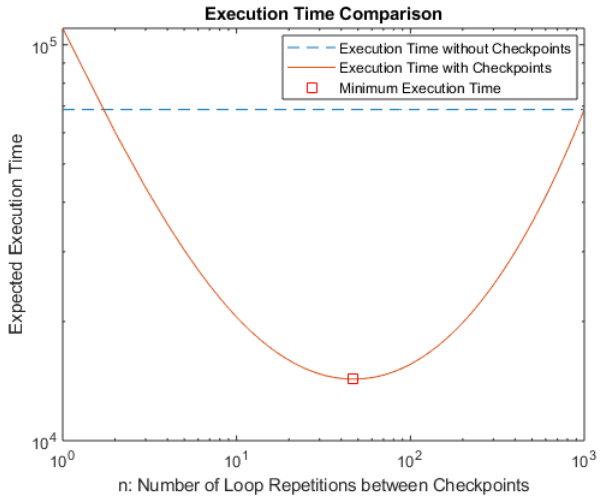
- If $r \geq 1$ we set $n = I(r)$,
- If $r < 1$, we set $n = I(\frac{1}{r})$.

To illustrate these results, some numerical examples are provided in order to show the effect of the checkpoint interval n (expressed in terms of the number of loop repetitions between checkpoints) on the expected execution time and the total energy consumption of a software application that operates in the presence of failures. In order to differentiate the effect of computation time and energy consumption, we use n^o to represent the checkpoint interval that minimizes the total computation time, while n^+ referers to the optimum checkpoint interval that minimizes the total energy consumption. Note that in the preceding analysis, n^o can be obtained by setting $\alpha = 1,\ \beta = 0$, while n^+ is obtained by setting $\alpha = 0,\ \beta = 1$.

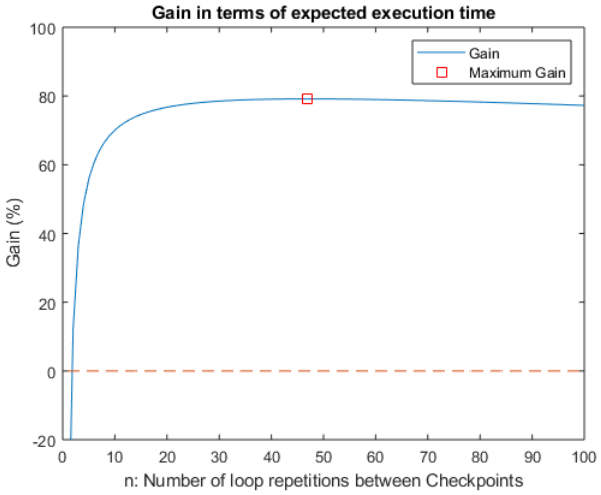
In the following examples we consider the case of a program with a single loop in which checkpoints are established at the beginning (or at the end) of the loop. We consider the cases of a small, medium, large, and very large program, comprised of $Y = 10^4,\ 10^5,\ 10^6,\ 10^7$ instructions, respectively. The expected execution time of the same program with and without the adoption of the ALCR mechanism is calculated and the corresponding optimization problem is solved numerically. The parameter values that we use are:

$$\begin{aligned}
&B_0^e = 500,\ B_1^e = 0,\ B_0^c = 10^5,\ B1c = 0,\ c^c = 1 \\
&c^e = 10^{-5},\ b_0^e = 100, b_1^e = 10,\ b_0^c = 100,\ b_1^c = 10 \\
&g = 5 \times 10^{-6},\ L = 100,\ N = 10^{-4}.
\end{aligned}$$

In Figure 1, the example of a small software program (i.e., $Y = 10^4$) is considered. Figure 1a compares the expected execution time of the application with and without the ALCR mechanism for different values of n, while Figure 1b shows the expected $Gain$ in terms of expected execution time for different values of n. The values that correspond to the optimum checkpoint interval n^o are marked within a rectangle.

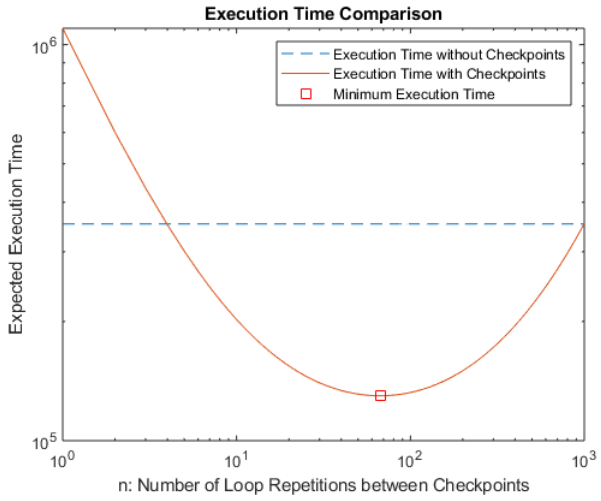


(a)

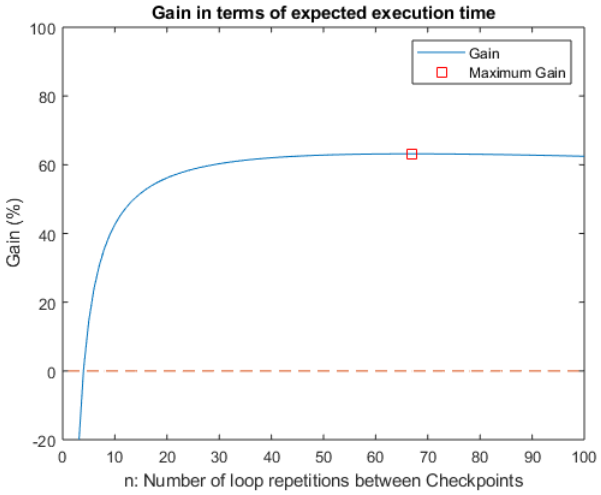


(b)

Fig. 1: The case of a small software program (i.e., $Y = 10^4$): (a) Expected execution time comparison (logarithmic axes) (b) Expected execution time gain.



(a)



(b)

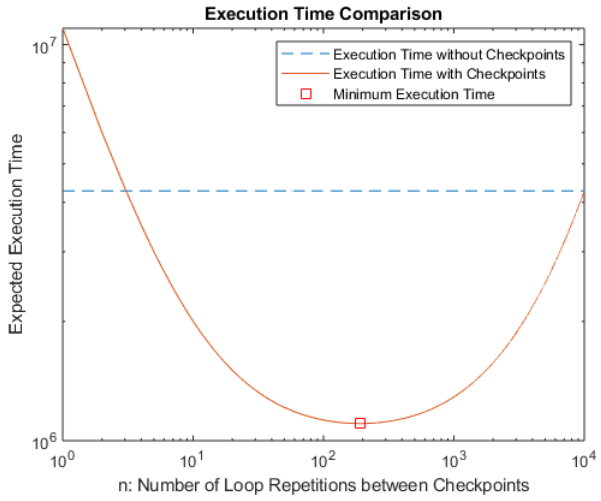
Fig. 2: The case of a software program of medium size (i.e., $Y = 10^5$): (a) Expected execution time comparison (logarithmic axes) (b) Expected execution time gain.

Figure 1 illustrates the fact that the optimum checkpoint interval n^o minimizes the overall execution time of the application and maximizes the overall expected Gain. From Figure 1 it is clear that the ALCR mechanism will not reduce the expected execution time of a given software application unless the checkpoint interval is optimally selected. Indeed, for some poorly chosen values of n, the expected execution time of the application with checkpointing is higher than the expected execution time of the same application without checkpoints.

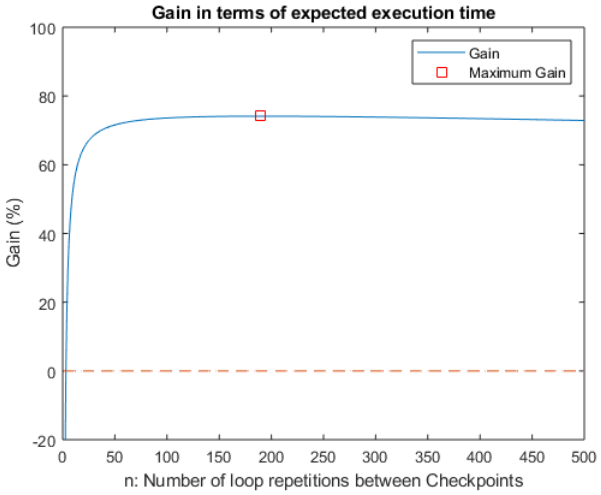
Similar observations can be made for software with longer loops in Figures 2, 3 and 4. This emphasizes the importance of setting n to be close or at n^o, when there is a need for minimizing the execution time of the program.

The examples of Figures 1, 2, 3 and 4 show that a significant reduction in the execution time of a software application can be achieved by the ALCR mechanism, if the checkpoint interval is selected to be at, or close to, the optimum n^o. In these examples the Gain, ranges from 64% to 80%. However, suboptimal values of the checkpoint interval will lead to a smaller Gain or even to an average execution time which is larger than when ALCR is not used. Indeed, the checkpoint interval should not be selected arbitrarily and must be tuned to a value at, or close to, the optimum n^o.

Still there is a relationship between calculations for n^o and n^+. However, we must have in mind that optimum checkpoint interval will be different regarding energy consumption and execution time. Figure 5 shows how they correspond to each other. More specifically, Figure 5a shows how execution time changes when we want to use optimal checkpoint interval calculated for energy consumption. Similarly Figure 5b shows how energy consumption changes when we want to use the checkpoint interval that optimizes execution time.

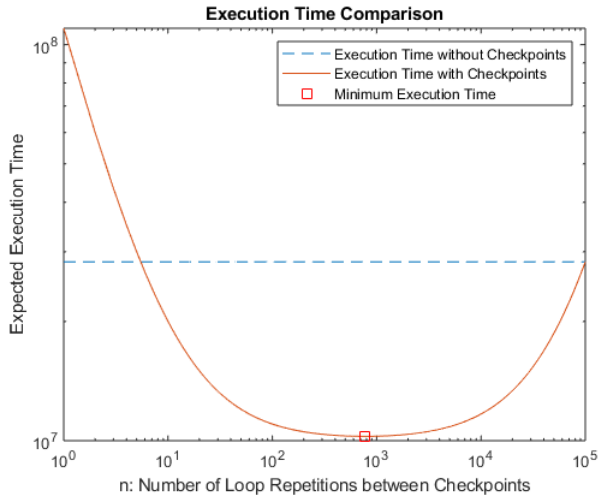


(a)

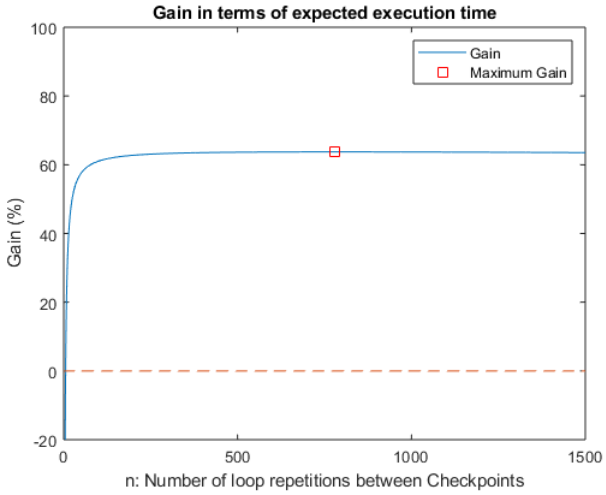


(b)

Fig. 3: The case of a large software program (i.e., $Y = 10^6$): (a) Expected execution time comparison (logarithmic axes) (b) Expected execution time gain.



(a)



(b)

Fig. 4: The case of a very large software program (i.e., $Y = 10^7$): (a) Expected execution time comparison (logarithmic axes) (b) Expected execution time gain.

A. *Impact of g and B on the Optimum Checkpoint Interval*

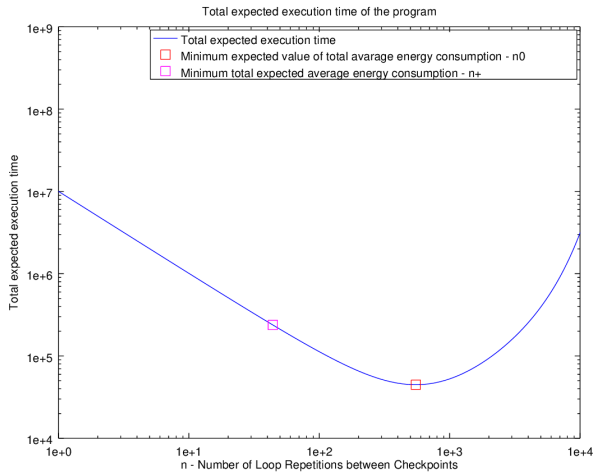
The optimum checkpoint interval n^o is expected to be influenced both by the probability of failure $g = 1 - a$, and by the cost of checkpointing $B^c = B_0^c$. In Figure **??**, the optimum checkpoint interval n^o is plotted against the probability of failure g, for three different cases of checkpointing cost B^c. Four different examples are provided, corresponding to a sample software program of small, medium, large, and very large size. In fact, the same cases of programs that were investigated in Section IV were considered in this section.

From the different graphs in Figure 6 and Figure 7, we notice that the same behavior is observed regarding the impact that the values of B^c and g have on the optimum checkpoint interval, regardlessof program size. Indeed for a given checkpointing cost B^c, the higher the probability of failure g, the lower the optimum checkpoint interval n^o. Conversely, for a specific probability of failure g, a higher cost of a single checkpoint B^c eads to a larger optimum checkpoint interval n^o.

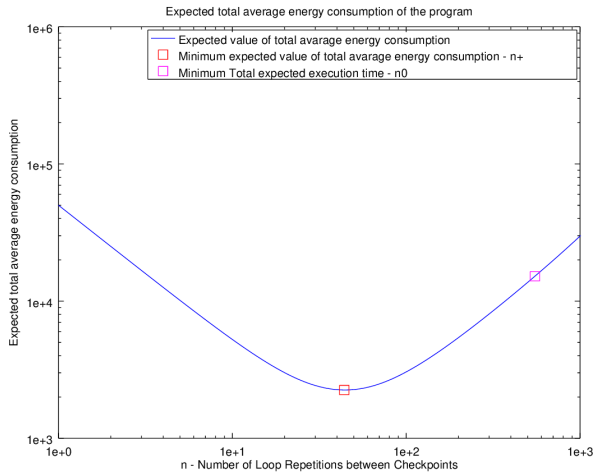
These observations are highly intuitive, since frequent checkpointing should be applied when the probability of failure is high, while checkpoints should be generated less frequently when the checkpointing cost is high. The same observations hold for the case of the optimum checkpoint interval n^+ that minimizes the total expected energy consumption of the program.

V. Conclusions

Checkpoints are widely used in databases, operating systems, and in high performance computing. They allow a system to recover from failures whithout having to restart each program's execution from scratch every time a failure occurs. However checkpointing has costs both in additional time and energy, even when no failures occur.

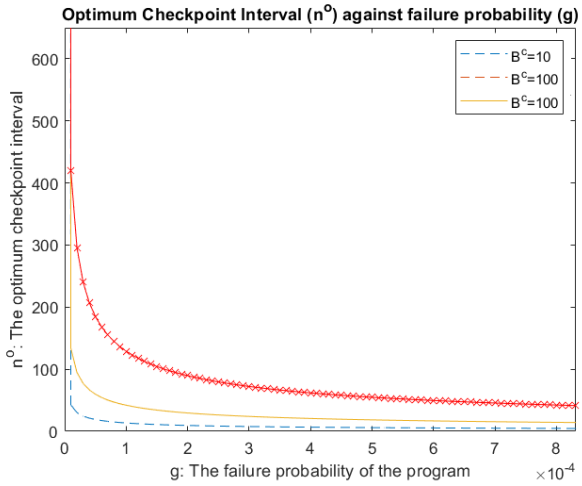


(a)

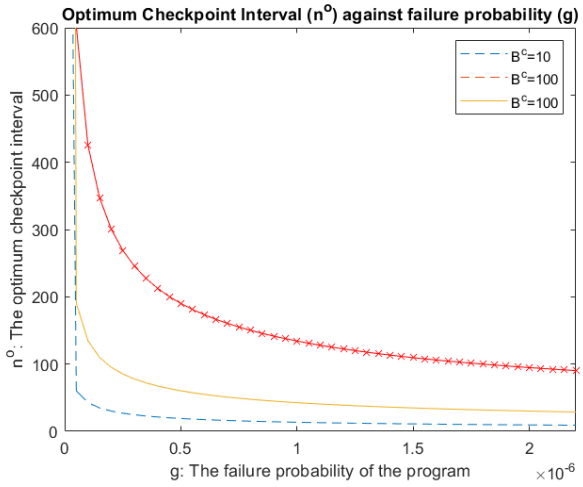


(b)

Fig. 5: The case of a large software program (i.e., $M = 10^6$): (a) Expected execution time with highlighted n^+. (b) Expected energy consumption with highlighted n^o.



(a)



(b)

Fig. 6: The optimum checkpoint interval n^o against the probability of failure g for different cases of checkpointing cost B^c, for a program of small (a) size with $Y = 10^4$, and a medium sized program in (b) with $Y = 10^5$.

Thus, this paper has analyzed the choice of optimum checkpoint intervals both from the perspective of energy costs and costs in execution time. Statring from first principles we have derived the optimum checkpoint for long running programs and detailed the analysis for programs with a long running outer loop. Explicit analytic results have been derived with closed form expressions and have been illustrated with numerical examples.

Future work will consider nested program structures and ways of linking checkpointing and program structure in a useful and intuitive manner, similar to what is done in this paper for programs witha large single loop.

REFERENCES

[1] B. Randell, "System Structure for Software Fault Tolerance," *Science*, no. 2, pp. 1–18, 1975. [Online]. Available: http://portal.acm.org/citation.cfm?doid=800027.808467

[2] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," *Parallel Process Letters*, vol. 28, no. 10, pp. 91–108, 1993.

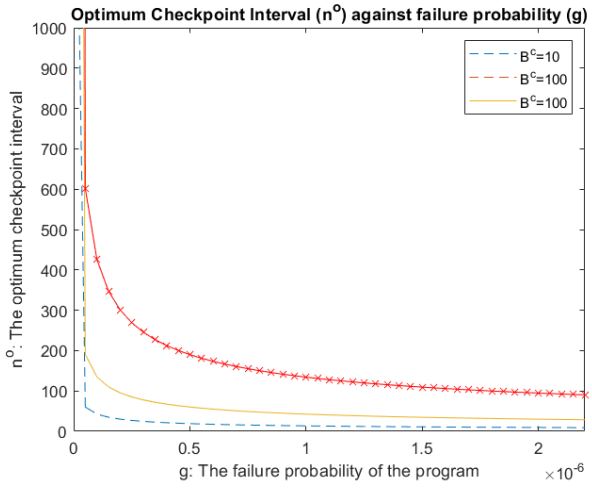
[3] G. Zheng, L. Shi, and L. V. Kale, "Ftc-charm++: an in-memory checkpoint-based fault tolerant runtime for charm++ and mpi," in *2004 IEEE international Conference on Cluster Computing*, September 2004, pp. 93–103.

[4] G. L. Stavrinides and H. D. Karatza, "The impact of checkpointing interval selection on the scheduling performance of real-time fine-grained parallel applications in SaaS clouds under various failure probabilities," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 12, p. e4288, 2018.

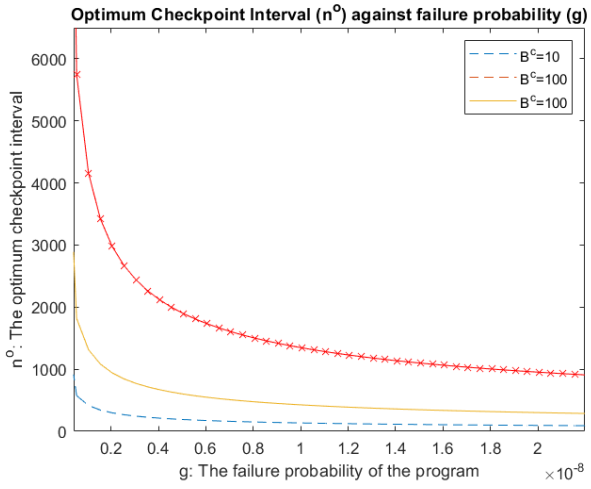
[5] D. Dauwe, R. Jhaveri, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Optimizing checkpoint intervals for reduced energy use in exascale systems," in *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2017, pp. 1–8.

[6] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[7] S. Vadhiyar and J. Dongarra, "Srs – a framework for developing malleable and migratable parallel software," *Parallel Processing Letters*, vol. 13, no. 2, pp. 291–312, 2003.



(a)



(b)

Fig. 7: The optimum checkpoint interval n^o against the probability of failure g for different cases of checkpointing cost B^c, for a large program a with $Y = 10^6$, and a very large program (b) with $Y = 10^7$.

[8] J. Mehnert-Spahn et al., "Architecture of the xtreemos grid checkpointing service," in *EuroPar 2009, LNCS 5704*. Springer, Cham, 2009, pp. 429–441.

[9] S. Agrawal, R. Garg, M. S. Gupta, and J. E. Moreira, "Adaptive incremental checkpointing for massively parallel systems," in *ICS '04: Proceedings of the 18th Annual International Conference on Supercomputing*. ACM, 2004, p. 277–286.

[10] A. Moody, et al., "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[11] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: transparent checkpointing under unix," *Technical Report UT-CS-94-242, Department of Computer Science, University of Tennessee*, 1994.

[12] J. Duell, "The design and implementation of berkeley lab's linux checkpoint/restart," April 2005. [Online]. Available: http://www.nersc.gov/research/FTG/checkpoint/reports.html

[13] J. B. M. Litzkow, T. Tannenbaum, and M. Livny, "Checkpoint and migration of unix processes in the condor distributed processing system," *Technical Report, University of Wisconsin, Madison*, no. 1346, 1997.

[14] M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Transact Comput. Syst.*, vol. 3, no. 1, p. 63–75, 1985.

[15] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proceedings of the 11th symposium on reliable distributed systems*, October 1992, p. 147–154.

[16] J. C. Sancho et al., "On the feasibility of incremental checkpointing for scientific computing," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings,*. IEEEXplore, 2004.

[17] M. Siavvas et al., "Static analysis-based approaches for secure software development," in *Security in Computer and Information Sciences. Communications in Computer and Information Science*, E. Gelenbe et al., Ed., vol. 821. Springer, Cham, 2018, pp. 142–157. [Online]. Available: http://link.springer.com/10.1007/978-3-319-95189-8{_}13

[18] R. Arora, "ITALC : Interactive Tool for Application - Level Checkpointing," *Proceedings of the Fourth International Workshop on HPC User Support Tools*, 2017.

[19] F. Shahzad, J. Thies, and G. Wellein, "CRAFT: A library for easier application-level Checkpoint/Restart and Automatic Fault Tolerance," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[20] N. Losada, M. J. Martín, G. Rodríguez, and P. Gonzalez, "Portable application-level checkpointing for hybrid MPI-OpenMP applications," *Procedia Computer Science*, vol. 80, pp. 19–29, 2016.

[21] G. Rodríguez, M. J. Martín, P. González, J. Touriño, and R. Doallo, "CPPC: a compiler-assisted tool for portable checkpointing of message-passing applications," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 749–766, 2010. [Online]. Available: http://dx.doi.org/10.1002/cpe.1541

[22] E. Gelenbe, "A model on information renewal by the method of multiple test points," *Avtomatika i Telemekhanika*, no. 4, pp. 142–151, 1979.

[23] S. K. Tripathi, D. Finkel, and E. Gelenbe, "Load sharing in distributed systems with failures," *Acta informatica*, vol. 25, no. 6, pp. 677–689, 1988.

[24] E. Gelenbe, D. Finkel, and S. K. Tripathi, "Availability of a distributed computer system with failures," *Acta Informatica*, vol. 23, no. 6, pp. 643–655, 1986.

[25] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Commun. ACM*, vol. 17, no. 9, pp. 530–531, Sep. 1974. [Online]. Available: http://doi.acm.org/10.1145/361147.361115

[26] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Commun. ACM*, vol. 21, no. 6, pp. 493–499, Jun. 1978. [Online]. Available: http://doi.acm.org/10.1145/359511.359531

[27] E. Gelenbe, "A model of roll-back recovery with multiple checkpoints," in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 251–255. [Online]. Available: http://dl.acm.org/citation.cfm?id=800253.807684

[28] E. Gelenbe and M. Hernández, "Optimum checkpoints with age dependent failures," *Acta Informatica*, vol. 27, no. 6, pp. 519–531, May 1990. [Online]. Available: https://doi.org/10.1007/BF00277388

[29] E. Gelenbe, "On the optimum checkpoint interval," *J. ACM*, vol. 26, no. 2, pp. 259–270, Apr. 1979. [Online]. Available: http://doi.acm.org/10.1145/322123.322131

[30] M. Siavvas and E. Gelenbe, "Optimum interval for application-level checkpoints," in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. IEEE, 2019, pp. 145–150.

[31] ——, "Optimum checkpoints for programs with loops," *Simulation Modelling Practice and Theory*, vol. 97, p. 101951, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X1930084X

[32] S. Dobson, et al., "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, 2006.

[33] A. Berl, et al., "Energy-efficient cloud computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[34] E. Gelenbe, "Energy packet networks: adaptive energy management for the cloud," in *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, 2012, pp. 1–5.

[35] R. Buyya, et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–38, 2019.

[36] B. Pernici, et al., "What is can do for environmental sustainability: a report from caise'11 panel on green and sustainable is," *Communications of the Association for Information Systems*, vol. 30, no. 1, 2012.

[37] E. Gelenbe and Y. Caseau, "The impact of information technology on energy consumption and carbon emissions," *Ubiquity*, vol. 2015, no. June, pp. 1–15, 2015.
[38] J. H. Lambert, "Observationes variae in mathesin puram," *Acta Helveticae Physico-Mathematico-Anatomico-Botanico-Medica*, vol. III, 1758.
[39] L. Euler, "De serie lambertina plurimisque eius insignibus proprietatibus," *Acta Acad. Scient. Petropol.*, vol. 2, pp. 29–51, 1783.
[40] G. Pólya and G. Szegö, *Aufgaben und Lehrsätze der Analysis*. Springer-Verlag, 1925.
[41] J. 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X04002213