Optimum Checkpoints for Time and Energy

Erol Gelenbe, Paweł Boryszko, Miltiadis Siavvas and Joanna Domanska

***Abstract*—We study programs which operate in the presence of possible failures and which must be restarted from the beginning after each failure. In such systems checkpointsare introduced to reduce the large costs of program restarts when failures occur. Here we suggest that checkpoints should be introduced in a manner which assures effective reliability, while reducing both the computational overhead as much as possible, but also to save energy. We compute the total average program execution time in the presence of checkoints so as to limit the re-execution time of the program from the most recent checkpoint. We also study the total energy cnsumption of the program under the same conditions, and formulate an optimization problem to minimize a wighted sum of both average computation time and energy. This approach is placed in the context of Application Level Checkpointing and Restart (ALCR). We then focus on checkpoints placed at the beginning of a loop, and derive the optimum placement of checkpoints to minimize a weighted combination of the program's execution time and energy consumption. Numerical results are presented to illustrate the analysis. Finally we describe a software tool with a graphical interface that has been designed to assist a system designer in choosing the optimum checkpoint for a given program as a function of different failure rates and other parameters.**

***Index Terms*—Optimum checkpoints; Program loops; Software reliability; Energy saving; Application Level Checkpoints and Restart (ALCR); Time optimization; Toolbox; User interface**

I. Introduction

Reliability has long been important to many applications, including for long-running software that performs computationally expensive or critical tasks [1]. Indeed, a single failure may lead to the need to re-execute a large number of instructions, resulting in significant overhead, reduction in effective performance performance and increase in energy consumption. The lack of reliability can also have dire consequences for time-critical software programs that have to provide their results within a specific time frame [2]–[4], as well as for smaller applications that have to ensure energy efficiency of their operation [5]. For all such applications, fault tolerance mechanisms that avoid the complete program re-execution in case of failures have long been studied [6], [7].

In particular, the Checkpoint and Restart (CR) technique [8] is widely used to keep a sequence of safe copies (or checkpoints) of the program execution state so that the program may be restarted from a recent checkpoint so as to avoid re-executing the whole program in case of failures. Checkpointing schemes are also popular in high performance computing systems [9]–[12], and have also been implemented in operating systems such as Unix or Linux [13]–[15], and to insure the consistency of distributed systems [16]–[18]. Multiple level checkpointing introduced in [19], [20] was also recently studied in [21].

The Application-level Checkpoint and Restart (ALCR) technique [22], [23] is an efficient approach to checkpointing which requires a relatively small memory footprint which is limited to a single application. However, implementing checkpoints requires programming skills, as well as expertise in selecting the locations in the application where additional checkpointing code will be inserted, and a judicious choice of the checkpoint intervals. Existing ALCR tools and libraries facilitate the programming aspects by helping to insert checkpoints within ong-running loops [24], [25], but typically do not offer guidelines to determine the interval betweeen checkpoints. Nevertheless checkpoint optimization applications using ALCR have been recently studied [26], [27].

While placing checkpoints increases the overhead for both execution time and energy for program execution, it can help save both time and power consumption during the recovery from when failures. This trade-off has been studies in several papers over the years using analytic models for transaction processing systems [28]–[30] and techniques that anlalyze a system's robustness under failures or attacks [31]–[34].

The checkpoint interval has often been used to maximize a system's availability for transaction processing systems or databases [35]–[39]. In addition, energy consumption minimization for computer systems is now recognized as a valid issue [40]–[44] with regard to the sustainability of information technology [45], [46]. However despite the importance of energy consumption in systems for reasons of economic cost and sustainability [47], [48], less work has been undertaken on judicious checkpointing to reduce the energy that is consumed by a system [49].

In this paper we take a common approach to checkpointing so as to save both the energy consumed by a program and its effective execution time. Using first principles, we construct a probability model that predicts the total average execution plus energy consumption of a program. Turning specifically to the case of checpointing with ALCR, we compute the corresponding quantities for a program built around a long repetitive loop. The analysis offers a way to compute the optimum checkpoint interval that minimizes the overall average cost, which we illustrate with several examples. Finally we present a software tool to implement these results, allowing the end user to select checkpoint intervals for ALCR equipped programs, without having to delve deeply into the underlying theory.

This research was supported by the European Commission through the H2020 SDK4ED Project under Grant Agreement No. 780572. The contents of this paper represent the opinion of the authors, and do not engage the responsibility of the European Commission.

Erol Gelenbe is with the Institute of Theoretical & Applied Computer Science, IITIS-PAN, 44-100 Gliwice, PL and the Université Côte d'Azur.

Paweł Boryszko and Joanna Domanska are with the Institute of Theoretical & Applied Computer Science, IITIS-PAN, 44-100 Gliwice, PL.

Miltiadis Siavvas is with the Information Technologies Institute, Centre for Research and Technology Hellas, 6th km Harilaou - Thermi, Thessaloniki 57001, GR

II. A SINGLE LOOP PROGRAM WITH CHECKPOINTS

Suupose that some program runs y_n instructions starting from its $(n-1)$th untill its nth checkpoint. Here y_n does not include the repetitions of instruction execution due to failuers, or the additional code executed for recovering from failures. At the instant $t_n > 0$ when the program creates its $n-th$ checkpoint llet Y_n be the total number of instructions that the program up to t_n and Y_n does not include the instructons that were repeatedly executed due to checkpoints and failures, so that $Y_n = \sum_{i=1}^{n} y_i$

If $B^c(Y_n)$ is the time during needed for the creation of the $n-th$ checkpoint, which depends on memory occupied the program occupies, and can also depend on Y_n due to data generated by the program, then $B^c(Y_n) = B_0^c + B_1^c Y_n$ with constants $B_0^c > 0$ and $B_1^c \geq 0$. When a failure occurs after the program has completed y instructions following a checkpoint, $b^c(Y_n, y)$ is the time required to start the program from the most recent checkpoint.

Thus the program successfully executes $y \leq Y_{n+1} - Y_n$ instructions after the most recent checkpoint and before the $(n+1)$th checkpoint, where $b^c(Y_n, y) = b_0^c + b_1^c y$ with constants $b_0^c > 0,\ b_1^c \geq 0$, depending on the total number of instructions executed since the last checkpoint, i.e.

- The time $B^c(Y_n)$ that is needed for establishing the n-th checkpoint will depend on the total number of instructions Y_n executed since the program was launched, where $B^c(Y_n) = B_0^c + B_1^c(Y_n)$,
- While the duration $b^c(Y_n, y)$ that is needed for failure recovery posterior to the nth checkpoint, including related reloading the state of the system after a failure, depends on $y <\leq Y_{n+1} - Y_n$, where $b^c(Y_n, y) \equiv b^c(y) = b_0^c + b_1^c y$.

If the energy consumption for creating the nth checkpoint is $B^e(Y_n)$, while $b^e(y)$ is the consumed energy for failure recovery when the total executed instructions is $Y_n + y\ leq Y_{n+1}$, we have $B^e(Y_n) = B_0^e + B_1^e Y_n$ where $b^e(y) = b_0^e + b_1^e y$ and $B_0^e > 0,\ B_1^e \geq 0,\ b_0^e > 0,\ b_1^e \geq 0$.

Calling $\alpha,\ \beta > 0$ the positive constants representing the relative cost of computation and energy, we characterize the joint total cost of time and energy through the quantities:

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

III. FIXED CHECKPOINT INTERVALS

Age dependent checkpoints [39] have been shown to reduce the overall cost of checkpointing and failure recovery, when the failure probability can increase with time, practical checkpointing generally makes the simple choice of periodically carrying out a checkpoint each time the program successfully executes a fixed number of instructions $y_n = y$. We will now proceed in this mmaner, and assume that checkpoints are installed after $Y_1 = y,\ Y_2 = 2y,\ ..\ Y_n = ny$ instructions are executed. We then need to select the optimum y, or the optimum n, that minimizes the composite cost that combines time and energy. If the program ends when $Y = Ny$ instructions are successfully executed, we do not need the $(N+1)$th checkpoint, and the first checkpoint is installed priot to executing the first instruction.

We formulate our problem for a program that executes Y successfully, and we select y or N, so that the total overhead in computation and energy consumption due to failure recovery and checkpoints is minimized.

$C^c(y)$, the total expected execution time including all restarts due to failures, starting from the most recent checkpoint, for an average execution time per instruction c and failure probability per instruction $(1-a)$, is then::

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

Indeed, with probability a^y a failure does not occur during the y instructions, leading to an execution time of $c^c y$ time units. However, with probability $(1 - a^y)$ one failure or more will occur in the y instructions. The first failure that occurs prvokes the restart of the program which takes time b_0^c, and we add to it the time $C^c(y)$ which includes the needed to restart and re-execute the program to execute y instructions after possible future failures.

Including the execution time and additonal work needed per executed instruction until occurance of the next failure, introduces the term $(c^c + b_1^c)$ multiplied by x and the probability that the failure occurs at instruction x, i.e. $a^{x-1}.a$, the whole being summed over x between 1 to y. Using

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

results in:

$$C^c(y) = b_0^c[a^{-y} - 1] + \frac{c^c + b_1^c}{1-a}[a^{-y} - 1] - b_1^c y. \tag{2}$$

The total expected energy consumption $C^e(y)$ for executing a number of instructions y after the most recent checkpoint, is also obtained in the same manner:

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

where c^e is the average energy consumption per executed instruction. therefore:

$$\begin{aligned} C(y) &= \alpha C^c(y) + \beta C^e(y), \\ &= b_0[a^{-y} - 1] + \frac{c + b_1}{1-a}[a^{-y} - 1] - b_1 y. \end{aligned} \tag{4}$$

IV. THE TOTAL COST

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:

$$K_N(y) = NB_0 + \sum_{i=1}^{N} iyB_1 + NC(y), \quad (5)$$

$$= N[B_0 + C(y)] + \frac{N(N+1)}{2} yB_1. \quad (6)$$

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

$$\kappa_N(y) \equiv \frac{K_N(y)}{Ny},$$
$$= \frac{B_0 + C(y)}{y} + \frac{(N+1)B_1}{2},$$
$$= \frac{B_0 + C(y) + \frac{B_1 Y}{2}}{y} + \frac{B_1}{2}, \quad (7)$$

since $Y = Ny$. Defining:

$$B = B_0 + \frac{B_1 Y}{2}, \quad (8)$$

we seek the value of y which yields the minimum of (7) , and compute its first derivative:

$$\frac{d\kappa_N}{dy} = \frac{y\frac{dC(y)}{dy} - [B + C(y)]}{y^2} . \quad (9)$$

Setting the derivative to zero we have:

$$0 = \frac{y\frac{dC(y)}{dy} - [B + C(y)]}{y^2}. \quad (10)$$

Then using:

$$A = b_0 + \frac{c + b_1}{1 - a}, \quad (11)$$

we obtain the expression for $y*$, the value of y which optimizes $\kappa_N(y)$:

$$0 = y^* \frac{dC(y)}{dy}|_{y=y^*} - [B + C(y^*)],$$
$$0 = A[-\frac{y^* \ln a}{a^{y^*}} - \frac{1}{a^{y^*}}] + A - B,$$
$$or\ y^* \ln a + 1 = a^{-y^*}][1 - \frac{B}{A}] . \quad (12)$$

If $B = A$ then the unique solution to (12) exists and $y^* = -[\ln a]^{-1}$ 0 since $a < 1$. If $B < A$, again (12) has a solution $y^* > 0$ since the equatons LHS decreases linearly from 1, while the RHS increases for $y^* > 1$, so that the RHS and LHS will intersect at some value y^*. If $B > A$ the RHS decreases for $y^* > 1$ and will intersect with the LHS at some point.

A. Showing that y^* is a minimum

To check whether y^* is a minimum, we derive:

$$\frac{d^2\kappa_N(y)}{dy^2} = \frac{y^3 C''(y) - 2y^2 C'(y) + 2y[B + C]}{y^4}, \quad (13)$$

where $C'(y)$, $C''(y)$ ate the first two derivatives of $C(y)$ with respect to y. At y^* we know that $y^* C'(y^*) = B + C(y^*)$, and write:

$$\frac{d^2\kappa_N(y)}{dy^2}|_{y=y^*} = \frac{C''(y)}{y}|_{y=y^*}, \quad (14)$$

so that to examine the sign of $C''(y^*)$ we obtain:

$$C''(y) = a^{-y}(\ln a)^2[b_0 + \frac{c + b_1}{1 - a}] - a^{-y} \ln a \frac{c + b_1}{(1 - a)^2}. \quad (15)$$

We see that $C''(y) > 0$, so that $\kappa_N(y)$ is minimized for $y = y^*$.

V. Programs with a Single Outer Loop

When software that manages a set of sensors and actuators, it will process data from all sensors, update some variables, and provide data to actuators and/or forward the data to some cloud server. This sequence may repeat indefinitely in this program with one outer loop. Thus the theoretical results we obtain can be applied to a program with an outer loop with L instructions, which is executed T times with $Y = LT$. Depending on y^*, we may be constrained to place checkpoints at the start of a loop or at every few starts of a loop, or several checkpoints may be inserted in each loop.

Thus we will first apply the results of Section III to compute the optimum number of instructions executed between successive checkpoints y^* that minimize a weighted sum of execution time and energy consumption, as indicated in the expression (12) where we write:

$$B = B_0 + B_1 L.T,\ A = b_0 + \frac{c + b_1}{1 - a}. \quad (16)$$

Let $I(x)$ be the integer that is closest to the real number x. Then we obtain the related value:

$$n = I(\frac{L}{y})\ if\ \frac{L}{y} \geq 1, \quad (17)$$
$$= I(\frac{y}{L}),\ if\ \frac{y}{L} > 1.$$

We will present examples in order to show the effect of y and hence n on the expected execution time and the total energy consumption of a program that runs in the presence of failures.

To differentiate the computation timee from energy consumption, $n = n^o$ will represent the value that minimizes the total computation time, while $n = n^+$ will referer to the value minimizing energy consumption. Thus in (**??**), we have n^o for $\alpha = 1,\ \beta = 0$, while n^+ corresponds to $\alpha = 0,\ \beta = 1$.

We consider a program with a single loop with checkpoints placed at the beginning (or at the end) of the loop. The parameters used are:

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

In Figure 1, a short program is taken with $Y = 10^4$) is considered. Its expected execution time is shown as a function of n.

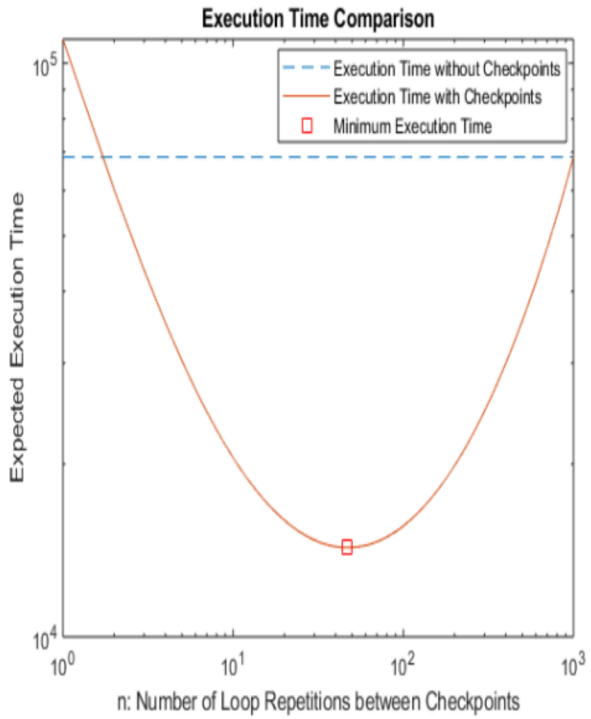

Fig. 1. Variation of the expected execution time with the number of checkpoints that are placed in a program with the total number of instructions executed being $Y = 10^4$.

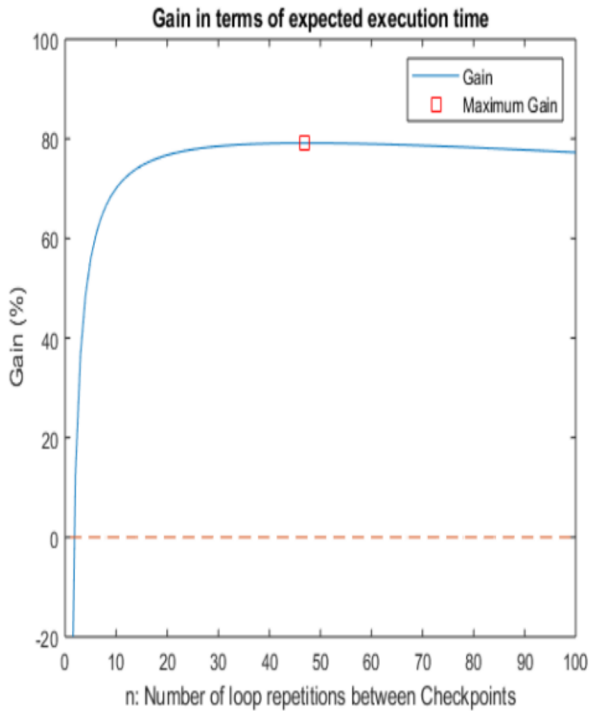

Fig. 2. Gain in the expected execution time he case for a small program with $Y = 10^4$.

We define the $Gain$ as the ratio of the exepected execution In Figure 2 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. It illustrates the fact that the optimum checkpoint interval n^o minimizes the overall execution time of the application and maximizes the overall expected Gain.

The relationship between n^o and n^+, the optimum checkpoint interval that minimizes computation time and energy, are investigated in Figure **??**. In Figure **??** we see how the execution time changes when we the use optimal checkpoint interval calculated for energy consumption. Similarly Figure **??** shows how energy consumption changes when we use the checkpoint interval that optimizes execution time.

VI. The Optimum Checkpointinging Tool

In addition to the theoretical work of solving the checkpointing problem analytically, it has proved very useful to build a software tool that can help system designers, who are not performance modeling experts, to install checkpoints in a way that minimize energy consumption and/or program execution time, by a judicious choice of the checkpoint interval.

Thus we have developed software consisting of a Dashboard front-end and back-end, that provides analysis, predictions and recommendations, in the form of micro-services written in Java and built into Docker Containers [50]. The recommendations can be triggered by the user, or retrieved from prior examples in a database built using MongoDB [51].

A graphical interface, which we call the "Dashboard", presents itself to the user and displays the analysis in the form of several display pages. This interface is based on the JavaScript library React [52] with the free user interface KIT [**?**] for building responsive websites and applications [53].

One of these components is the Dependability Toolbox which consists of three key features. Two of them: (i) Quantitive Security Assessment and (ii) Vulnerability Prediction, are responsible for maintaining the security of the applications. The third is responsible for maintaining the reliability of applications and is based on the assessing (iii) The optimal checkpoint interval between successive checkpoints. The calculations and predictions are triggered by the user, or can be retrieved from the database built on top of MongoDB.

One of the pages presents the results for the optimal checkpoint interval for a given program in terms of execution time and energy consumption. It is done by identifying the most demanding loop in the prgram and recommending the optimal interval between checkpoints to provide the lowest possible execution time or energy consumption. Calculations can also be done concerning both of them at the same time with user specified by relative importance.

The user of platform will first login to her account and choose the project from the projects page. Next, she can go to the proper page presenting the optimal checkpoint interval for her application as presented in Figure 3. It is possible for the user to see the name of the program and to perform the analysis using the proper button if there have been changes in the program. The date of the last analysis is also shown on the dash page.

After the data is taken from the new analysis or the database, the user can see the results divided into two sections. One of them presents the outcome for execution time in Figure 4, and the other for energy consumption in Figure 5. Both of them share a similar template. It presents the relevant data to the user the in the form of the plot, from which the user can see how the specified metric, the average computation time and the average energy consumption of the program varies concerning the number of loop repetitions between checkpoints. This is powered by the Plotly library and is located on the left of the section. To the right, there is shown to us two smaller sections. They present the minimum value from the set and also the corresponding value of the distance between checkpoints in the form of loop repetitions. Below we can also see the detailed table of the values if the user prefers to choose differently than optimal value.

To perform the analysis we first need provide the back-end of the dependability toolbox with specific parameters in the form of the JSON shown in **Listing 1** through the available API.

Listing 1. Request body example

```
{   "ProgramType":"OptimalCheckpoints",
"g":"0.000005",
"B0e":"0.00000059",
"B0c":"0.00000347",
"L":"2826.0",
"ce":"0.000000000445",
"cc":"0.000000000074231",
"b0c":"0.000000077",
"b1c":"0.0000000007",
"b0e":"0.000000367",
"b1e":"0.00000000367",
"N":"200.0",
"alfa":"0.0",
"beta":"1.0",
"B1e":"0.0",
"B1c":"0.0",
"Y":"19782.0",
"history_data":"1.0",
"project_name":"Neurasmus",
"username":"userName"}
```

This consists mainly of parameters needed to perform the analysis for selecting the checkpoint interval of the program, and parameters that distinguish this particular program which may be selected from data stored in the database. After retrieving the data through the back-end the new analysis is performed. The outcome of the calculations is then transferred to the database to store the newest data, and exhibited on the particular dashboard page that is dedicated to optimal checkpoint interval analysis. Again the data exchanged between back-end and front-end are sent using the JSON format as presented in the **Listing 2**.

Listing 2. Response body example

```
{ "_id" : { "$oid" : "5f0ea3d0d60180001080cbcc" },
"energyConsumptionTable" : {
"columns" : [
{ "field" : "y", "label" : "Energy consumption" },
{ "field" : "x", "label" : "Number of loop repetitions
    between checkpoints" }],
"rows" : [
{ "x" : 1, "y" : 1.308E-9 },
{ "x" : 2, "y" : 7.0E-10 }, ...
{ "x" : 200, "y" : 3.658E-9 }] },
"calculationSummary" : {
"nPlusIndex" : 14,
"nStarValue" : 6.05E-10,
"nStarIndex" : 3,
"nPlusValue" : 2.44E-10 },
"executionTimeOverNumberOfInstructions" : [{
"mode" : "line",
"margin" : { "r" : 20, "b" : 50, "t" : 50, "l" : 20 },
"marker" : { "color" : "red" },
"x" : [1, 2, ... 200],
"y" : [6.85E-10, 6.1E-10, ... 1.9462E-8],
"type" : "scatter",
"autosize" : "true" }],
"executionTimeTable" : {
"columns" : [
{ "field" : "y", "label" : "Execution time" },
{ "field" : "x", "label" : "Number of loop repetitions
    between checkpoints" }],
"rows" : [
{ "x" : 1, "y" : 6.85E-10 },
{ "x" : 2, "y" : 6.1E-10 }, ...
{ "x" : 200, "y" : 1.9462E-8 }] },
"project_name" : "Neurasmus",
"energyConsumptionOverNumberOfInstructions" : [{
"mode" : "line",
"margin" : { "r" : 20, "b" : 50, "t" : 50, "l" : 20 },
"marker" : { "color" : "red" },
"x" : [1, 2, ... 200],
"y" : [1.308E-9, 7.0E-10, ... 3.658E-9],
"type" : "scatter",
"autosize" : "true" }],
"username" : "userName",
"timestamp" : "July 15, 2020 6:36 AM"
}
```

VII. Conclusions

Checkpoints are widely used in databases, operating systems, high performance computing, and in long-running programs for real-time applications. They allow a system or a program to recover from failures whithout having to restart execution from scratch each time a failure occurs. However checkpointing has costs both in additional time and energy, even when no failures occur. Indeed, the placement of checkpoints themselves consumes system resources.

Thus, this paper has analyzed the choice of optimum checkpoint intervals both from the perspective of energy costs and in terms of execution time, innovating with respect to previous work that has focused principally on execution time. Starting from first principles we have derived the optimum checkpoint for long running programs. We have also detailed the analysis for programs with a long running outer loop, which is common to many applications which deal with real-time systems. Explicit analytic results have been derived with closed form expressions and have been illustrated with numerical examples. We have also presented the design and operation of a tool that incorporates these results.

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

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

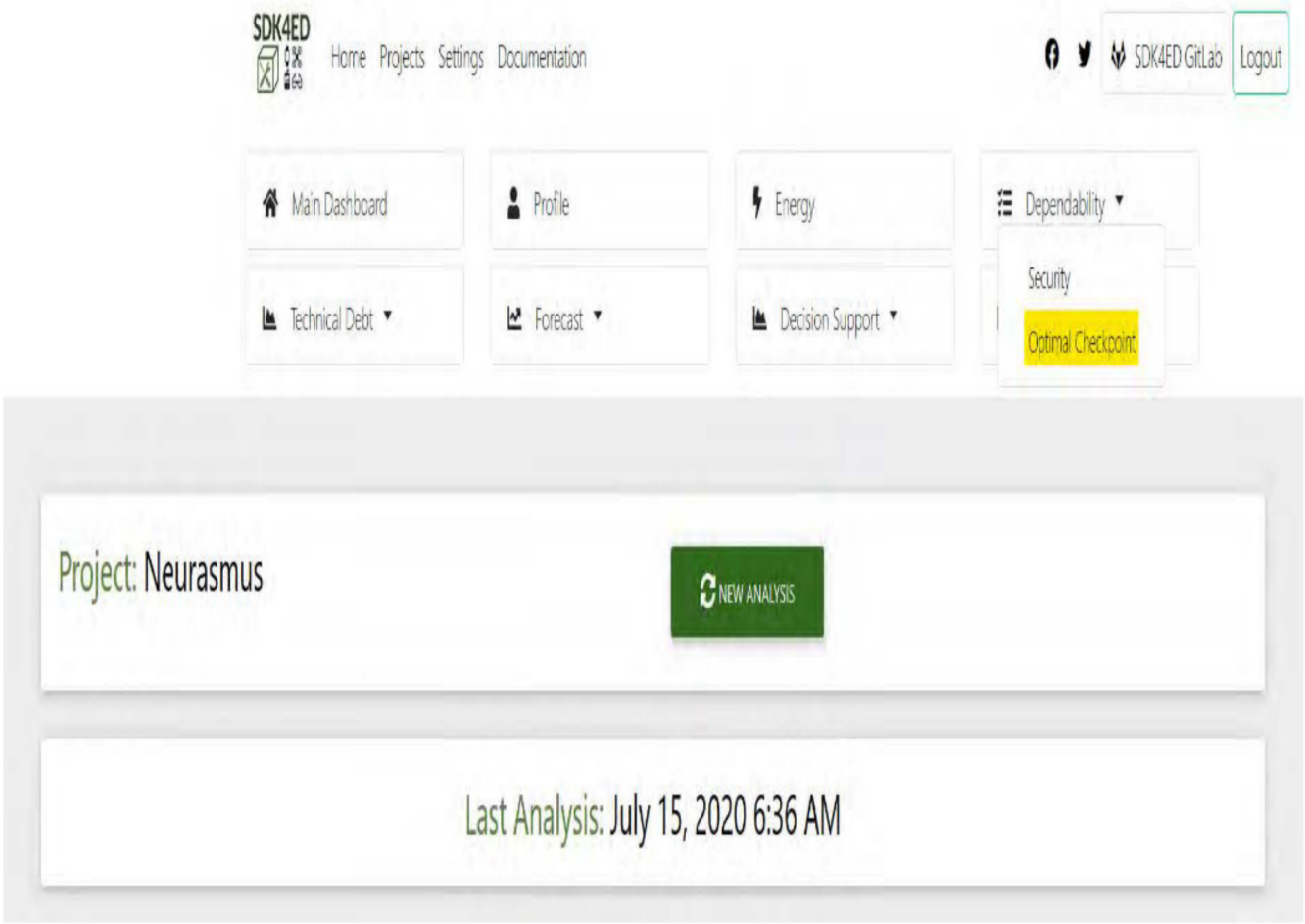

Fig. 3. Accessing the Optimal Checkpoint Dashpage

[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 et al., "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] K. M. Chandy and C. V. Ramamoorthy, "Rollback and recovery strategies for computer programs," *IEEE Trans. Computers*, vol. 21, no. 6, pp. 546–556, 1972. [Online]. Available: https://doi.org/10.1109/TC.1972.5009007

[7] 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.

[8] K. M. Chandy, "A survey of analytic models of rollback and recovery strategies," *IEEE Computer*, vol. 8, no. 5, pp. 40–47, 1975. [Online]. Available: https://doi.org/10.1109/C-M.1975.218955

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] E. Gelenbe, "A model of roll-back recovery with multiple checkpoints," in *Proceedings of the 2Nd International Conference on Software*

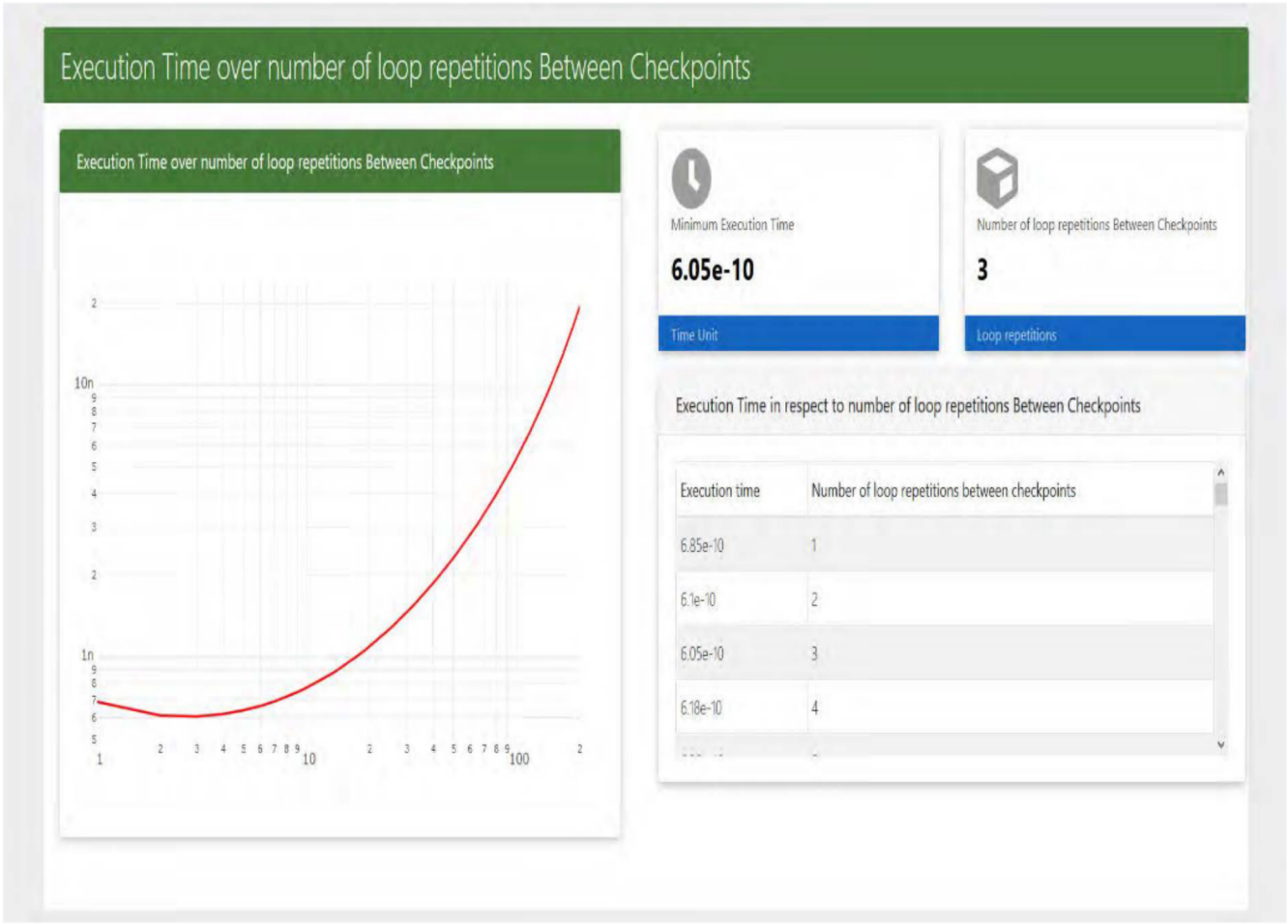

Fig. 4. Part of page for execution time

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

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

[21] A. Benoit et al., "Towards optimal multi-level checkpointing," *IEEE Transactions on Computers*, vol. 66, pp. 1212–1226, Ju;y 2017.

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

[23] 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.

[24] 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.

[25] G. Rodríguez et al., "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

[26] 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.

[27] ——, "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

[28] E. Gelenbe et al., "Availability of a distributed computer system with failures," *Acta Informatica*, vol. 23, no. 6, pp. 643–655, 1986.

[29] E. Gelenbe, R. Lent, and M. Douratsos, "Choosing a local or remote cloud," in *2012 Second Symposium on Network Cloud Computing and Applications*. IEEE, 2012, pp. 25–30.

[30] N. Bajunaid and D. Menasce, "Efficient modeling and optimizing of checkpointing in concurrent component-based software systems," *Journal of Systems and Software*, vol. 139, pp. 1–13, May 2018.

[31] S. K. Tripathi et al., "Load sharing in distributed systems with failures," *Acta informatica*, vol. 25, no. 6, pp. 677–689, 1988.

[32] E. Gelenbe and T. Koçak, "Area-based results for mine detection," *IEEE transactions on geoscience and remote sensing*, vol. 38, no. 1, pp. 12–24, 2000.

[33] O. Brun et al., "Deep learning with dense random neural network for detecting attacks against iot-connected home environments."

[34] S. Evmorfos et al., "Neural network architectures for the detection of SYN flood attacks in iot systems," in *PETRA '20: The 13th PErvasive Technologies Related to Assistive Environments Conference, Corfu, Greece, June 30 - July 3, 2020*. ACM, 2020, pp. 69:1–69:4. [Online]. Available: https://doi.org/10.1145/3389189.3398000

[35] 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

[36] 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

[37] 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

[38] A. Chesnais et al., "On the modeling of parallel access to shared data," *Communications of the ACM*, vol. 26, no. 3, pp. 196–202, 1983.

[39] E. Gelenbe and M. Hernández, "Optimum checkpoints with age

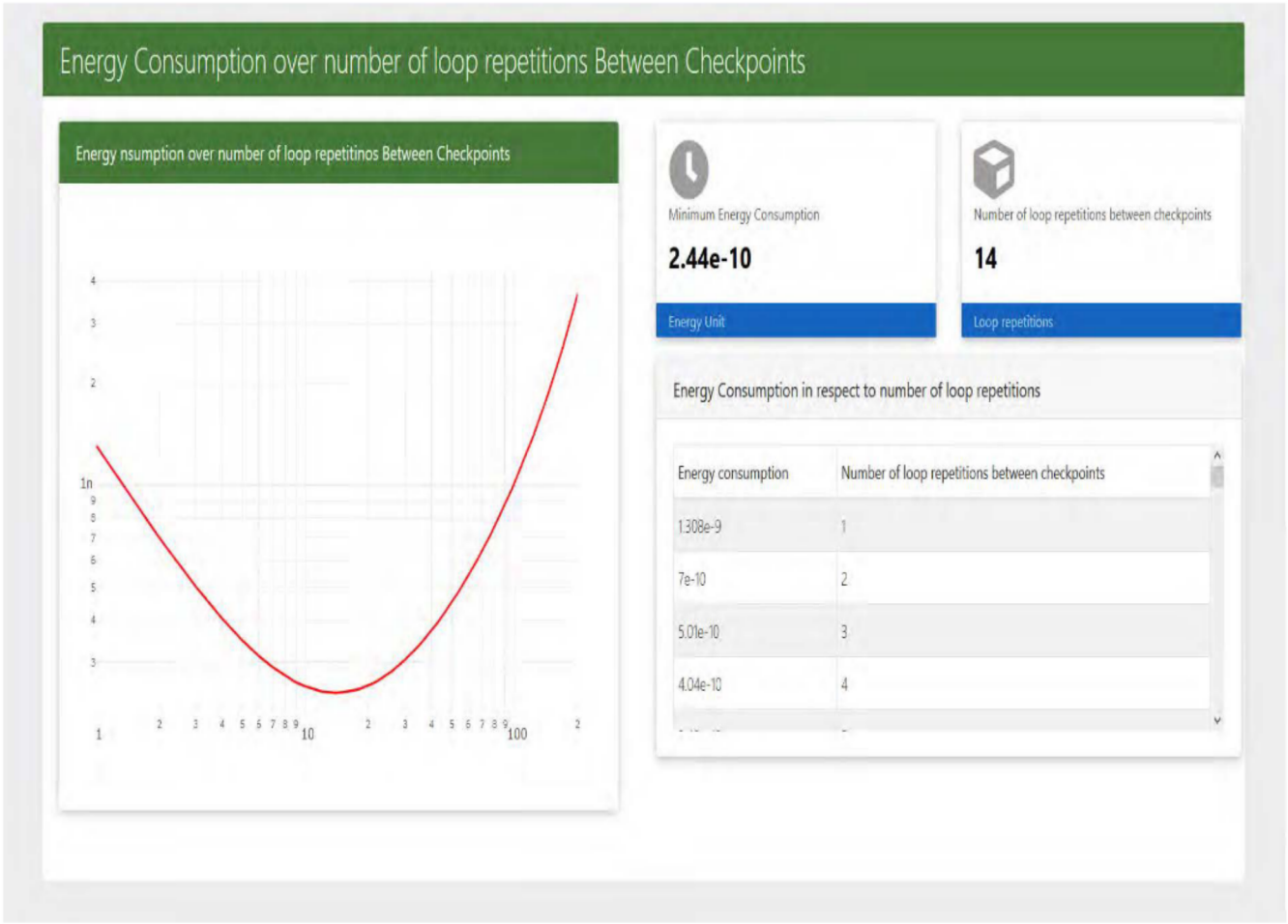

Fig. 5. Part of page for energy consumption

dependent failures," *Acta Informatica*, vol. 27, no. 6, pp. 519–531, May 1990. [Online]. Available: https://doi.org/10.1007/BF00277388

[40] 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.

[41] E. Gelenbe, "Energy packet networks: adaptive energy management for the cloud," in *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*. ACM, 2012, pp. 1–5. [Online]. Available: https://doi.org/10.1145/2168697.2168698

[42] F. Almeida et al., "Energy monitoring as an essential building block towards sustainable ultrascale systems," *Sustain. Comput. Informatics Syst.*, vol. 17, pp. 27–42, 2018. [Online]. Available: https://doi.org/10.1016/j.suscom.2017.10.013

[43] G. L. Stavrinides and H. D. Karatza, "The impact of workload variability on the energy efficiency of large-scale heterogeneous distributed systems," *Simul. Model. Pract. Theory*, vol. 89, pp. 135–143, 2018. [Online]. Available: https://doi.org/10.1016/j.simpat.2018.09.013

[44] 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.

[45] E. Gelenbe and Y. Caseau, "The impact of information technology on energy consumption and carbon emissions," *Ubiquity*, no. June, pp. 1–15, 2015. [Online]. Available: https://doi.org/10.1145/2755977

[46] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, qos-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations," *Future Gener. Comput. Syst.*, vol. 96, pp. 216–226, 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.02.019

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

[48] 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.

[49] G. Aupy et al., "Energy-aware algorithms for task graph scheduling, replica placement and checkpoint strategies," in *Handbook on Data Centers*, S. U. Khan and A. Y. Zomaya, Eds. Springer, 2015, pp. 37–80. [Online]. Available: https://doi.org/10.1007/978-1-4939-2092-1_2

[50] "Docker Container," https://www.docker.com/resources/what-container.

[51] "MongoDB Database," https://www.mongodb.com/.

[52] "React - JavaScript Library," https://reactjs.org/.

[53] "Material Design for Bootstrap," https://mdbootstrap.com/.