Revista Brasileira de Computação Aplicada, July, 2020

DOI: 10.5335/rbca.v12i2.10295
Vol. 12, Nº 2, pp. 16–27
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ORIGINAL PAPER

Evaluating the impact of a coordinated checkpointing in distributed data streams processing systems using discrete event simulation

Matheus Bernardelli de Moraes [1] and André Leon Sampaio Gradvohl [1]

[1]School of Technology – University of Campinas

*matheuzmoraes@gmail.com; gradvohl@ft.unicamp.br

Received: 2019-11-29. **Revised:** 2020-05-19. **Accepted:** 2020-06-06.

Abstract

Coordinated Checkpointing is a fault-tolerance strategy proposed for Data Streams Processing systems, which handles a continuous, potentially unbounded flow of data under Quality of Service requirements. Although traditional in large-scale distributed systems, there is a lack of study on how a Coordinated Checkpointing may impact the stream processing in both failure-free and failure-prone environments, especially considering the inherent requirement of analyzing and processing data in real-time. This paper presents a study that used a discrete simulation model to investigate the impacts of the Coordinated Checkpoint fault tolerance strategy on a Data Stream Processing System. The results show Coordinated Checkpointing should be avoided since it critically impacts the stream processing and the real-time analyzes of data, increasing latency up to 120%, and discarding up 95% of the processing window during a global checkpoint when a rollback-recovery is required.

Keywords: Data Streams; Fault-Tolerance; Coordinated Checkpoint; Rollback-Recovery; Simulation Analysis

Resumo

Checkpoint Coordenado é uma estrategía de tolerância a falhas proposta para Sistemas de Processamento de Fluxos de Dados, que processam um fluxo de dados contínuo e potencialmente infinito dentro dos requerimentos da Qualidade de Serviço. Embora tradicional em sistemas distribuídos de larga-escala, existe uma falta de estudos em como o *Checkpoint* Coordenado pode afetar o processamento de fluxos de dados em cenarios com e sem falhas, especialmente considerando a necessidade inerente de analise e processamento em tempo real dos dados nesse tipo de sistema. Esse trabalho apresenta um estudo que usou um modelo de simulaçação discreto para investigar os impactos da estratégia de tolerância a falhas de *Checkpoint* Coordenado em um Sistema de Processamento de Fluxos de Dados. Os resultados demonstram que o *Checkpoint* Coordenado deve ser evitado, visto que afeta criticamente o processamento de fluxos e a análise em tempo real dos dados, aumentando a latência em até 120% e descartando até 95% dos dados da janela de processamento durante um *checkpoint* global quando um *rollback-recovery* é necessário.

Palavras-Chave: Fluxos de Dados; Tolerância a Falhas; Checkpoint Coordenado; Recuperação por Retorno; Análise de Simulação

1 Introduction

Data Stream Processing (DaSP) systems are a computing paradigm for online analysis of data streams processed under Quality of Service (QoS) requirements (de Matteis and Mencagli, 2017). These streams are potentially unbounded data transmitted at high volume and high velocities. Some of them require real-time processing and analysis, such as disaster management, network attack and anomaly detection, financial market, trend analysis, social media, web analytics, Internet of Things (IoT), operational infrastructure monitoring, and online advertising (de Assunção et al., 2018, Gradvohl et al., 2014).

DaSP systems have to process data streams uninterruptedly to provide real-time analysis. The system must be fault-tolerant to achieve this level of dependability. One of the proposed fault-tolerance used for DaSP systems is the Checkpoint Rollback-Recovery. It consists of periodically saving the application's state to restart from the last safe state in case of a system failure. The checkpoint can be coordinated (synchronous) or uncoordinated (asynchronous). In the coordinated checkpoint, all components take a checkpoint at the same time. In turn, in the uncoordinated checkpoint, each component decides when to perform its checkpoint (Casanova et al., 2015).

Although most DaSP systems run in distributed process architectures, where the checkpoint-rollback-recovery strategy is intensely studied (Levy et al., 2014, Oldfield et al., 2007, Moody et al., 2010, Monnet et al., 2004), there is a lack of studies about the impact of this checkpoint strategy on DaSP systems. Nevertheless, practical evaluation of fault-tolerance mechanisms in large-scale applications such as DaSP systems is challenging.

At the hardware level, the challenges include the requirement to study machines that are either larger than those currently available or have hypothetical architectures. Other challenges in this level include the study of more advanced machines, which are not accessible yet; and the lack of analytical models to predict performance and compare to other results accurately (Levy et al., 2014).

Besides, at the application level, the system expects uncertainties, such as changes in arrival rate, arrival distribution, and others since data stream processing is potentially unbounded. Therefore, tests concerning failure issues and how the adopted fault-tolerance strategy interferes in stream processing are relevant as well.

Simulations are quite useful for performance analysis in parallel and distributed programs (Albertsson, 2006, Hoefler et al., 2010, Tikotekar et al., 2007) as well as in large and extreme-scale applications (Levy et al., 2014, Ferreira et al., 2011, Mubarak et al., 2012, Böhm and Engelmann, 2011). Besides, a simulation holds several benefits such as providing a risk-free environment, high accuracy compared to analytic models and the ability to handle uncertainty scenarios such as failure occurrences.

Therefore, we propose a simple discrete event simulation model built on ARENA simulation software to verify the impact of the Coordinated Checkpoint Rollback-Recovery (CCRR) strategy on the DaSP systems. The primary goal of this work is to simulate different situations in both failure-free and failure-prone scenarios. Also, we provide an application-driven simulation model capable of evaluating different QoS metrics such as latency, throughput, and mean waiting time; and integrity metrics such as the amount of information loss and unprocessed tuples.

The specific contributions of this paper are the following:

- a simple and easy to use a discrete event simulation model of the Coordinated Checkpoint-Rollback-Recovery in Stream Processing Systems;
- an evaluation of our model's performance showing an error of less than 1% and 11% against analytic models for both failure-free and failure-prone environments;
- a simulation analysis showing that the Coordinated Checkpointing could be impracticable in failure-prone DaSP systems due to high information loss, an increase in latency and a decrease in throughput reaching 95%;
- two analytic models to predict information loss in failure-free and failure-prone environments using CCRR.

We organized the remaining of this paper as follows: Section 2 presents the fundamental concepts. Section 3 shows the related work; Section 4 introduces the proposed computational model; Section 5 compares the simulation results and the analytic models; Section 6 describes the experiments; Section 7 discusses the results; and, finally, Section 8 presents the conclusions.

2 Fundamental Concepts

There are different architectures for online data processing and analysis. However, most of them are multi-tiered systems with loosely coupled components combined to form a single processing framework. This organization improves maintainability, scalability, and availability (de Assunção et al., 2018).

The multi-tiered architecture of DaSP systems comprises different components. Fig. 1 shows an overview of these components. For instance, there are Data Sources responsible for data streams generation, such as RFID readers, wireless sensors, mobile devices and GPS, among others. Also, there are Data Collectors, for instance, network clients, JSON readers, protocol buffers, and others, to gather the data streams and transmits them to the stream processing engine. Messaging systems are generally present as well. Examples of such message systems are IoT hubs queuing systems and publish-subscribe messages that receive the stream and manages it (de Assunção et al., 2018).

There are also Stream Processing Engines that will effectively process the streams and Data Deliveries, such as Web Interfaces, Dashboards, RESTful APIs, which will receive the processed information. The architecture also requires Data Storage components, like relational databases, NoSQL databases, or in-memory storage. However, using all components are not mandatory, and an actual system may have only some of these features. The communication between

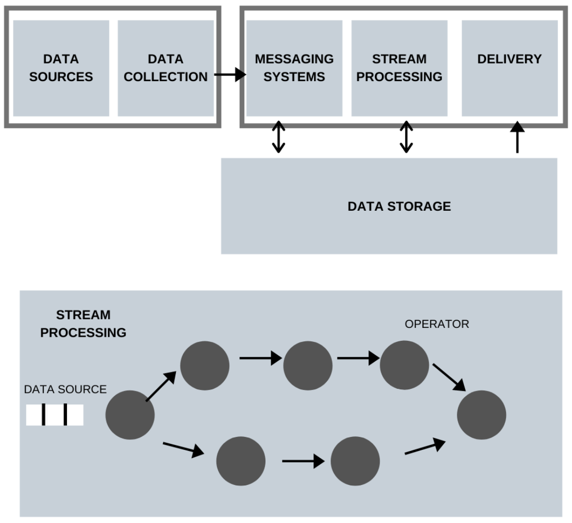


Figure 1: Basic architecture of a DaSP system.

components often uses TCP/IP protocols (Gradvohl et al., 2014).

The Stream Processing Engine uses several software components known as operators, running on processing nodes (hardware components). Each operator runs on a single node, although a single node can hold one or more operators. The engine connects the operators forming a directed acyclic graph (DAG), which we will refer to as a topology (Gradvohl, 2016). The operators are responsible for tuples processing and analyzing, and can execute a series of procedures such as data cleaning, classification and feature selection, among others.

We classify operators according to their ability to maintain their state, i. e., internal data structures, intermediary results, and tuples routing information, among others. We classify an operator as stateless if it does not gather or keep any information about the previously processed streams or the operator state. On the other hand, the output of the stateful operators depends on the processing of the previous streams and its previous state (Gradvohl, 2018).

Operators are components responsible for data stream processing. Beyond the requirement for real-time processing, data streams have other characteristics that distinguish them from traditional static data processing. They are potentially infinite, which makes them impractical for storing in the system's main memory; the system must analyze each tuple a limited number of times and discard them later to reduce the computational costs, to avoid queuing and offer a real-time response (Ramírez-Gallego et al., 2017).

Also, we formally define an input stream as a sequence of data elements $\{s_1, s_2, \ldots\}$, which each $s_i = (t_i, D_i)$, t_i is the time stamp, and $D_i = (d_1, d_2, \ldots)$ is the payload for each element i. In this paper, we consider s_i as a tuple. Second, we consider that the probabilistic distribution of the data may change over time. This phenomenon is well known and well studied in the data streams environment due to its non-stationary nature. We refer to this phenomenon as Concept Drift (Gama et al., 2014).

2.1 Coordinated Checkpointing

The system implements a coordinated or synchronous checkpoint by exchanging messages between the operators in a DaSP system. We can formally define a global checkpoint (or a snapshot) of a system composed by nodes $(n_1, n_2, n_3, \ldots, n_n)$ at an instant t as a storage of events at each n_i at the instant t and also a storage of the communications logs (send and receive messages) between operators in the instant t (Goswami and Sahu, 2005). Therefore, for global consistency, checkpoints are $(C_1^k, C_2^k, C_3^k, \ldots, C_n^k)$, where C_i^k is the k^{th} local checkpoint at node n_i.

Fig. 2 illustrates a simplified flowchart of the CCRR in DaSP systems. When the checkpoint interval expires, the model triggers the CCRR strategy. First, the strategy blocks all operators for stream processing, which results in information loss since the system discards all received tuples discarded. Then, the initiator sends a checkpoint request to the operators. All available operators reply to a message informing the initiator that they are active. If the system detects no failure, the initiator waits for all operators to perform their respective and local checkpoints. After this procedure, the initiator sends a message to all active operators informing that the system performed a global checkpoint successfully. This whole process is known as Commit Time.

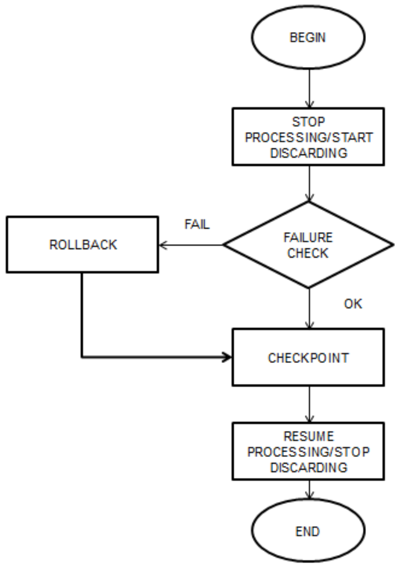

BEGIN
STOP PROCESSING/START DISCARDING
FAIL
ROLLBACK
FAILURE CHECK
OK
CHECKPOINT
RESUME PROCESSING/STOP DISCARDING
END

Figure 2: Flowchart of the Coordinated Checkpoint Rollback-Recovery strategy.

At the checkpoint request, if an operator does not respond, the initiator usually waits an extra time. If the initiator still receives no response, this indicates that a failure occurred. This process triggers the rollback recovery phase when the system waits until the node resumes and then recovers the last checkpoint of the failed operator. When the rollback-recovery process finishes, all operators take the checkpoint, and the initiator commits them to a stable storage device.

3 Related Work

In this section, we present works that approach stream processing simulators. Hoefler et al. (2010) divide simulators into different categories, such as application, application-communication, and architectures simulators. Application simulators focus on the performance of a given algorithm, a system, or an application. Users employ application-communications to simulate critical components of an application, such as its relation to other components presented in the topology. Finally, architecture simulators represent a detailed model of one or more components of a parallel architecture. The model presented in this paper is an application model.

Due to the high computational cost of a detailed simulation, simulations commonly focus on a limited group of components (Hoefler et al., 2010). A simulation model has to be accurate enough and yet avoid unnecessary features (Levy et al., 2014). Therefore, the work presented in the literature focuses on specific aspects of the distributed discrete event simulation.

Concerning failure tolerance aspects, Ferreira et al. (2011) has studied the benefits of the process replication as a primary fault-tolerance mechanism for large-scale distributed systems. They used different simulators to run the experiments. On the other hand, Levy et al. (2014) proposed a framework on the LogOPS simulator to evaluate the performance of the CCRR in large-scale systems.

On performance evaluation, Zheng et al. (2005) used BigSim as a simulation tool to develop a performance-modeling environment to predict performance issues on large parallel machines. In turn, Shchur and Shchur (2015) studied the benefits of using parallel discrete event simulation as a paradigm for large-scale modeling systems, including the requirement of analyzing important metrics such as scalability, CPU time, and storage issues.

For online processing, there are few works addressing simulations. For instance, CEPSim (Higashino et al., 2016) is a simulator for cloud-based systems that can model different DaSP systems by transforming them into user queries based on DAG representation. CEPSim allows some customizations of operators' execution, placement, and schedule while providing important metrics such as latency and throughput. However, CEPSim does not support fault-tolerance simulations.

In turn, Flow is a simulator primarily focused on the large-scale simulation of stream processing systems (Park et al., 2010). It is capable of working with millions of kernels and data flows, and the automatic parallelization of different models. As CEPSim, Flow also does not support fault-tolerance simulations. Table 1 presents a comparison table with CEPSim, Flow and our model.

Table 1: Comparison table of simulators for stream processing

Simulator	Evaluated metrics	Fault-Tolerance
CEPSim	Latency, Throughput, Execution Time, Memory Consumption	No
Flow	Hardware performances, Scalability	No
Our model	Latency, Throughput, Mean Waiting Time, Unprocessed Tuples, Information Loss	Yes

Concerning discrete event simulation software, there are many options available. For distributed systems, for instance, there are BigSim (Zheng et al., 2004), LAMMPS (Plimpton, 1995), xSim (Böhm and Engelmann, 2011) and LogGOPSim (Hoefler et al., 2010), among others. In turn, ARENA has embedded components such as resource allocation, queue management, and failure modules, which simplify the modeling of both DaSP system topology and the CCRR strategy. Researchers have already been using it for simulation of distributed systems (Christine and Emilie, 2005) and fault-tolerance strategies (Mehresh et al., 2010).

Therefore, although there are many works in discrete-event simulations of distributed systems, none of them addresses the specific and dynamic environment of the DaSP systems, except for CEPSim and Flow. Also, since both CEPSim and Flow do not support any fault-tolerance simulations, we find a lack of studies about the impact a fault-tolerance mechanism has on DaSP systems. This paper contributes to the proposition of a novel simulation model capable of performance and simulation analysis of the CCRR strategy in DaSP systems in both failure-free and failure-prone environments.

4 Computational Model

This section presents the simulation model and the input, control, and output parameters regarding our proposed approach.

4.1 System Model

Fig. 3 shows the simulation model. Since ARENA is a drag-and-drop simulation software, each component is a block that performs a specific computation. We modeled the data sources as *Create* blocks. The time in seconds (s) between arrivals follows a Normal Distribution of mean $\mu = 3.2 \times 10^{-3} s$ and standard deviation $\sigma = 5 \times 10^{-5} s$. Beyond the probabilistic distribution, the simulation user can set the number of entities per arrival, which defines the arrival rate. The default rate is 1, which is equivalent to 1250 tuples/s.

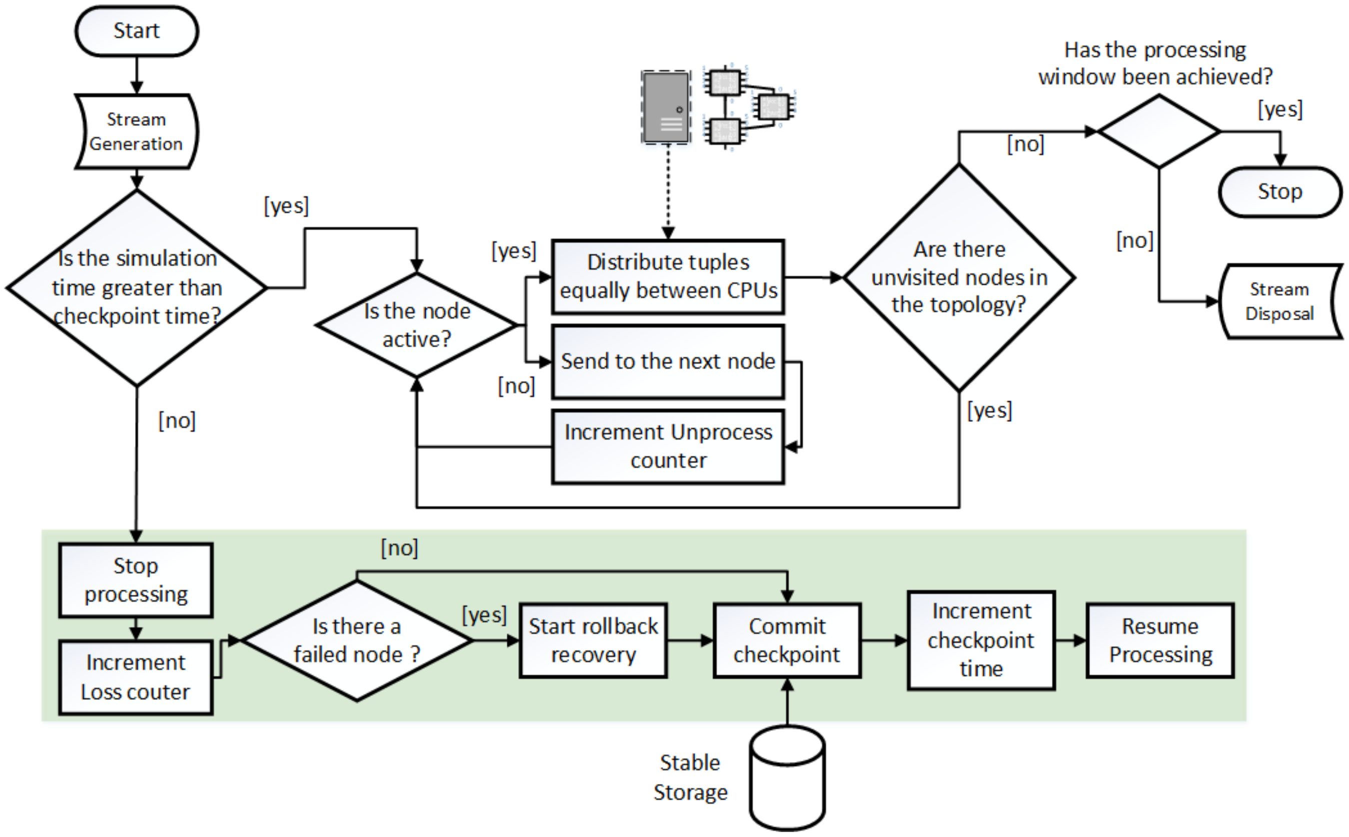


Figure 3: Proposed simulation model.

On the other hand, data collectors are the *Dispose* blocks. The stream-processing level sends the tuples, and the data collectors discard them from the simulation. It is important to observe that, since data streams are potentially infinite, the user must use a processing window to verify the system during a predetermined period. This period may be time-based when the system achieves a predefined amount of time, or tuple-based, calculated based on how many tuples the data collectors received. This work uses a tuple-based processing window by setting a termination condition on the execution setup with the number of tuples that the collectors successfully discarded.

At the stream-processing level, each operator receives tuples from the previous operator (except Operator 1, which receives tuples directly from data sources) through an Input Queue I and then sends them to the next operator throughout an Output Queue O. An operator communicates only with the next operator, except when it detects a failure. If it is the case, the model sends tuples to the next active operator. The *Decision* blocks before every node represent this condition, and the *record* blocks immediately classify all the tuples directly sent to the next operator as unprocessed.

Streams usually flow throughout the model. However, the stream-processing level receives the tuples, and a *Decision* block verifies if the simulation time is higher than the checkpoint expected time. In the affirmative case, it means the checkpoint time has expired, and the system has to take a global checkpoint. The model immediately stops the processing, and sends every received tuple to the loss area, increasing the counter for this metric.

Then, the model verifies if a node has failed. If this is the case, it triggers the recovery phase. A *Process* block models the recovery phase, which takes a constant user-applied variable of time (R) to recover. After the recovery, the initiator commits the checkpoint to the stable storage. The commit process is also a *Process* block, which takes a user-applied constant of time (δ) to execute. If it detects no failures, the system only performs the commit. Then, the model increments the checkpoint time, resume the processing and stops losing tuples. We modeled the failures as time-based on a Poisson distribution of mean M, and we can attach it to any node.

Fig. 4 shows the architecture level. There are four processing nodes, each one with three CPUs. There is only one operator in each processing node to simplify the model. We modeled the processing nodes as *resources* and the CPUs as *Process* blocks. They operate in a size, delay, and release procedure and process each tuple in a Normal Distribution of mean $\mu = 8 \times 10^{-5}s$ and standard deviation of $\sigma = 10^{-5}s$. Therefore, the proposed simulation model consists of four processing nodes. To increase the model scalability, a user can model more processing nodes by simply adding more *Process* blocks.

We can use several heuristics approaches to estimate the CPU time (Zheng et al., 2005). It can be a user-supplied expression, a suitable multiplier such as a scaling factor, a hardware performance counter to count floating-point, integer, and branch instructions on the simulation machine or a hardware simulator, which cycles a target machine processor. The proposed model uses a user-supplied expression since it is the less complex and highly flexible approach.

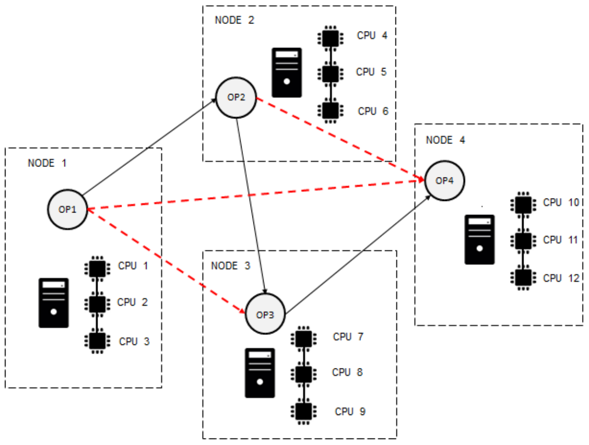


Figure 4: Sample DaSP architecture.

4.2 Input, control and output parameters

Fig. 5 shows a black box approach to the simulation model. Input parameters are entities generated and processed throughout the simulation. The model uses Control parameters to simulate distinct scenarios. Finally, output parameters are the metrics a simulation model intends to provide.

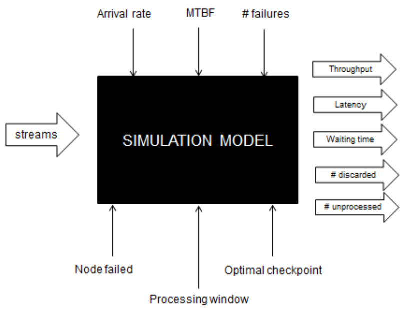


Figure 5: Simulation model black box.

Latencies are the main QoS parameters that a DaSP must attend. **System Latency** is the time a system takes to process and analyze a certain amount of data (Gradvohl, 2018). Therefore, the model requires low latency. There are other types of latency, such as Maximum peak latency, Post-peak latency, and Operator latency, which we do not address in this paper, but the model can measure them. In our model, the latency is equivalent to the simulation time since the model will stop when the system computes the number of tuples defined in the processing window.

Another QoS metric frequently observed in DaSP systems is the **throughput**, the rate of successfully processed tuples given a predetermined period (Gradvohl, 2016). In this paper, we use the seconds (s) as the adopted period. The simulation model requires high throughput.

Finally, we also account for the **mean waiting time** in the queue. Since data streams require real-time processing, failures, or the adopted fault-tolerance strategy must not substantially increase queuing time as it would increase both computational cost and latency.

Concerning integrity metrics, **unprocessed tuples** are the ones who did not pass through one or more operators. In DaSP systems, when a node fails, the system forwards the tuples that the failed operator would receive to the next active operator. This procedure is fundamental to maintain system availability. Considering that each operator may implement critical procedures (e. g., data cleaning, normalization, or classification), a high number of unprocessed tuples could lead to inaccurate decisions.

Information loss is also a crucial integrity metric. This metric computes the number of tuples discarded during the checkpoints. Critical information could have missed during this activity since coordinated checkpoint blocks all operators for stream processing. Besides the commit time, if a failure has occurred, the simulation will also account for the recovery time, which provokes an even severe situation.

All values set to *process* and *create* blocks were empirically defined to simulate the same arrival rate and processing time presented by Apache Storm, a real-world DaSP system, on the work presented by Chintapalli et al. (2016).

4.3 Limitations and Assumptions

Simulations are known as computationally expensive (Levy et al., 2014). In order to construct an efficient and accurate simulation model, we only modeled features that are relevant to the performance of the DaSP system and the adopted fault-tolerance strategy. Therefore, we made the following assumptions:

- The operators receive, process and send tuples based on a First-In-First-Out (FIFO) nature;
- Nodes work under the fail-stop model;
- We assume reliable message delivery; therefore, no message is lost;
- CPUs are identical.

Since the proposed simulation model is application-oriented, and we assume reliable message delivery, the simulation ignores failures in the network and the communication between operators. Besides, we do not directly address memory consumption.

5 Analytic Models

This section introduces the analytic models presented in the literature to predict execution time and the proposed analytic models to predict information loss, in both failure-free and failure-prone environments. In this section, we also compare the predicted with the obtained results in the simulation model.

5.1 Simulation time

Levy et al. (2014) proposes Eq. (1) for execution time prediction in failure-free environments.

$$T_W = T_S + \frac{T_S}{\tau} \times \delta \tag{1}$$

where T_W is the predicted execution time; T_S is the execution time without any fault-tolerance mechanisms; τ is the optimal checkpoint interval time; and δ is the commit time to the stable storage. For the cases where the CCRR shares the stable storage device, the authors propose the commit time expressed in Eq. (2).

$$\delta = \frac{N \times ||C_{\text{avg}}||}{\beta} \tag{2}$$

where N is the number of processing nodes; $||C_{\text{avg}}||$ is the average checkpoint size for each node, and β is the aggregate write bandwidth for the stable storage.

However, in failure-prone environments, we cannot use Eq. (1) since it does not consider failure occurrence. Therefore, we use Eq. (3) proposed by Daly (2006), which accounts for both failure occurrence and the required recovery time by using the mean time between failures (MTBF) and a constant R, described as follows:

$$T_{\text{fail}} = T_{\text{app}} + (k-1) \times \delta + k \times \left(\frac{\tau+\delta}{2} + R\right) \times \left(\frac{\tau+\delta}{M}\right) \tag{3}$$

In Eq. (3), M is the mean time between failures (MTBF); k is the number of performed checkpoints; and R is the node recovery time. According to Daly (2006), we assume $T_{\text{app}} = k \times \tau, k \in \mathbb{N}$.

Both (Eqs. (1) and (3)) use the variable τ, the optimal interval checkpoint time. Assuming $\delta \leq 2M$, we calculated this time using Eq. (4) as proposed by Daly (2006).

$$\tau = \left(\sqrt{2 \times \delta \times M}\right) \times \left[1 + \frac{1}{3} \times \sqrt{\frac{\delta}{2M}} + \frac{1}{9} \times \left(\frac{\delta}{2M}\right)\right] \tag{4}$$

5.2 Information loss

We can use Eq. (5) to predict information loss in a failure-free state.

$$\Omega = \epsilon \times (k \times \delta) \tag{5}$$

where Ω is the number of tuples lost; ϵ is the arrival rate; k the number of checkpoints; and δ the commit time.

However, Eq. (5) is not adequate for a failure-prone environment since it ignores failure occurrence. Therefore, we propose Eq. (6) to predict information loss in situations where failures occur.

$$\Omega = \epsilon \times (k \times \delta) + \epsilon \left(\left\lfloor \frac{T_{\text{fail}}}{M} + 1 \right\rfloor \times R \right) \tag{6}$$

where Ω is the number of tuples lost; ϵ is the arrival rate; k the number of checkpoints; δ the commit time; R is the recovery time; M is the MTBF; and T_{fail} is the execution time for a failure-prone environment.

On the other hand, unprocessed tuples are challenging to predict. We can calculate this metric using the exact time between a failure and the remaining time to the next checkpoint times the arrival rate. However, since researchers modeled them based on the MTBF and a predefined probabilistic distribution, values can change substantially even inside a simulation model. Therefore, using a simulation model is a practical approach to measure this metric.

Figs. 6 and 7 show the comparison between analytic models and the obtained results in the simulation. We considered the same values for each comparison, which were $M = 600s$, $R = 60s$, $\delta = 74s$ and $\epsilon = 5000$. The error for the latency prediction in failure-free environments was 1.5% and for failure-prone 10.5% in the worst case. The error for information loss prediction was less than 1% for failure-free environments and less than 11.6% for failure-prone environments. These results show that our model is accurate to simulate both latency and information loss.

6 Experimental Evaluation

In this section, we present four case studies applied to evaluate our proposed simulation model. We evaluated all cases on different arrival rates, and we replicated each experiment 10 times for each arrival rate. Therefore, we consider the mean for each value and a 10 million tuples processing window for all case studies.

Case 1 is the baseline test, a failure-free environment without the CCRR strategy. It is essential to verify the model performance running on a clean scenario to compare it to the remaining case studies. Since it is a failure-free environment, the only control parameter we used was the arrival rate.

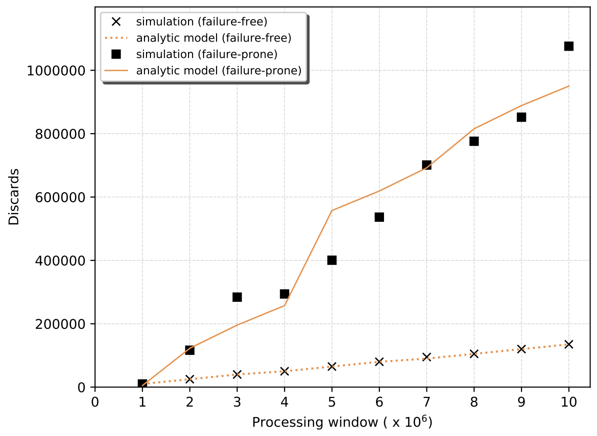


Figure 6: Information loss versus processing window size.

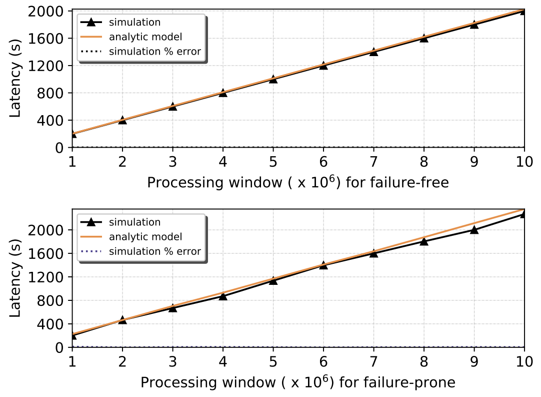


Figure 7: Latency versus processing window.

Case 2 is a failure-free environment with the implementation of the CCRR strategy. Therefore, no failure occurs. In this case, we want to verify how CCRR affects the system performance even if the system detects no failures. Concerning control parameters, we considered $\delta = 1s$, $M = 600s$ and the calculated checkpoint interval was $\tau = 74s$.

Case 3 is a failure-prone environment where a single node (Node 1) experiences a failure. The experiment relies on the investigation of the model performance in case of failures. We considered $\delta = 1s$, $M = 600s$, $R = 60s$ and the calculated checkpoint interval was $\tau = 74s$.

Case 4 is an emergency mode where two nodes (Node 1 and Node 3) fails at the same time. We increased the recovery time in 100%, and we reduced the MTBF in half to force more failures occurrences. Therefore, we considered $\delta = 1s$, $M = 300s$, $R = 120s$ and the calculated checkpoint interval was $\tau = 45s$.

7 Results and Discussion

Fig. 8a depicts the results for the latency metric. Results show that the increase in latency with the adoption of the CCRR strategy in failure-free environments is relatively low, with a maximum of 2%. A failure in a single node (Case 3) resulted in an increase in latency up to 10% compared to a failure-free environment (Case 2). For Case 4, the increased latency was up to 120% in the worst case.

Fig. 8b shows the results for the throughput metric. Given a certain arrival rate, it measures how much time the system takes to process and analyze all the received data from the first to the final processing node. Using the CCRR strategy also does not severely affect this metric in failure-free environments, with a decrease up to 2%. Case 3 showed a decrease up to 12%, and Case 4 showed a decrease up to 109%.

Therefore, evidence shows that the adoption of the CCRR strategy does not profoundly affect latency and throughput in failure-free environments. However, in emergencies, the CCRR strategy critically affects these metrics, reaching an increase of up to 120% in some cases. An increase of this magnitude could damage the real-time processing characteristics of a DaSP system.

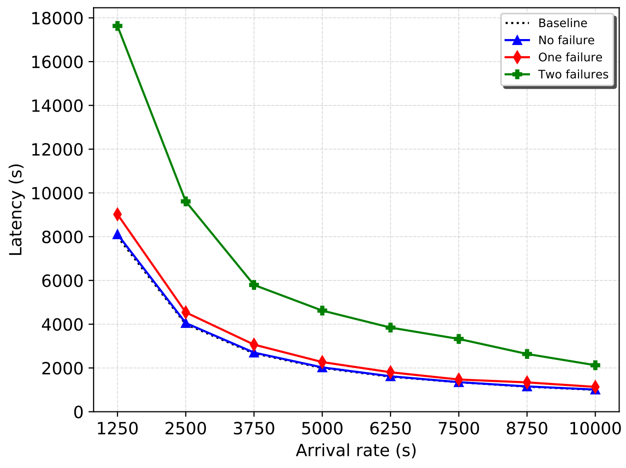
Tables 2 to 5 show the mean waiting time (in seconds) that a tuple in the queue waited for processing in each operator in the four studied cases. In Case 1 and Case 2, there was almost no difference between values, except for operators 3 and 4 in arrival rates 5000 and 6250. Therefore, evidence shows there is no substantial increase in mean waiting time in failure-free environments with the adoption of the CCRR strategy.

Table 2: Mean waiting time in seconds for Case 1. Dashed cells represent values so close to zero that could not be measured by the ARENA software.

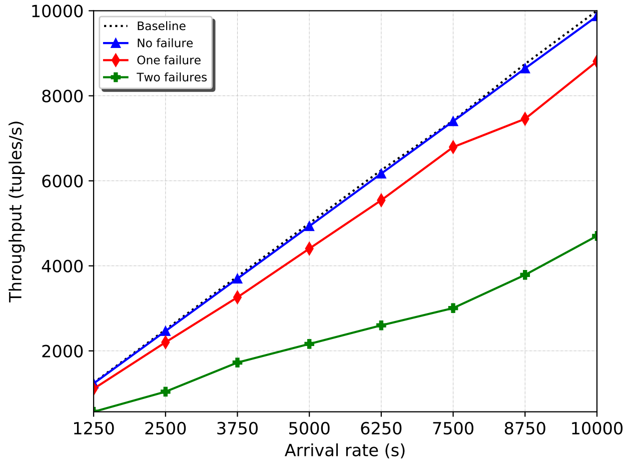
Arrival rate	Operator 1	Operator 2	Operator 3	Operator 4
1250	0.000288	–	–	–
2500	0.000180	0.000072	0.000108	0.000108
3750	0.000468	0.000288	0.000180	0.000108
5000	0.000612	0.000288	0.000324	0.000252
6250	0.000828	0.000288	0.000360	0.000324
7500	0.001080	0.000324	0.000396	0.000396
8750	0.001368	0.000324	0.000468	0.000432
10000	0.001692	0.000324	0.000468	0.000432

Table 3: Mean waiting time in seconds for Case 2. Dashed cells represent values so close to zero that could not be measured by the ARENA software.

Arrival rate	Operator 1	Operator 2	Operator 3	Operator 4
1250	0.000288	–	–	–
2500	0.000396	0.000108	0.000108	0.000108
3750	0.000468	0.000288	0.000144	0.000108
5000	0.000612	0.000288	0.000324	0.000180
6250	0.000828	0.000288	0.000360	0.000360
7500	0.001080	0.000324	0.000432	0.000360
8750	0.001368	0.000324	0.000468	0.000432
10000	0.001692	0.000324	0.000468	0.000432



(a) Latency.



(b) Throughput.

Figure 8: Latency and throughput versus arrival rate.

Table 4: Mean waiting time in seconds for Case 3. Dashed cells represent values so close to zero that could not be measured by the ARENA software.

Arrival rate	Operator 1	Operator 2	Operator 3	Operator 4
1250	0.000288	0.000288	–	–
2500	0.000396	0.000360	0.000108	0.000108
3750	0.000504	0.000396	0.000180	0.000108
5000	0.000612	0.000468	0.000360	0.000252
6250	0.000900	0.000576	0.000396	0.000360
7500	0.001152	0.000684	0.000396	0.000360
8750	0.001548	0.000756	0.000468	0.000432
10000	0.001908	0.000828	0.000504	0.000432

Table 5: Mean waiting time in seconds for Case 4. Dashed cells represent values so close to zero that could not be measured by the ARENA software.

Arrival rate	Operator 1	Operator 2	Operator 3	Operator 4
1250	0.000288	0.000288	–	–
2500	0.000396	0.000324	0.000108	0.000108
3750	0.000540	0.000396	0.000144	0.000108
5000	0.000684	0.000468	0.000360	0.000360
6250	0.001008	0.000576	0.000396	0.000360
7500	0.001296	0.000612	0.000468	0.000396
8750	0.001548	0.000720	0.000576	0.000432
10000	0.002016	0.000864	0.000540	0.000468

Concerning Case 3, there was an increase in time up to 12% for Operator 1, 121% for Operator 2, and 8% for Operator 3. For Operator 4, the results were almost the same, except for a 29% increase in time for 5000 tuples/s, when compared with the failure-free environments.

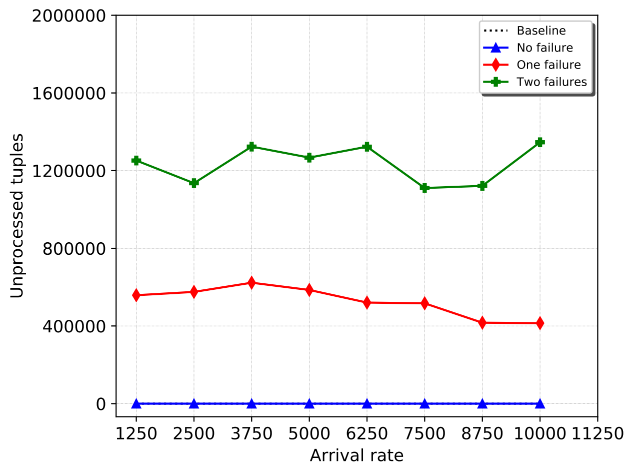
In Case 4, the increase was up to 27%, 125%, 14%, and 8% for operators 1, 2, 3, and 4, respectively. Therefore, it shows a relation between a failure (which we implemented in nodes 1 and 3 where operators 1 and 3 were running) and an increase in the mean waiting time on the next operators, especially on the closest one. However, although expressive, none of these increases were high enough to affect the stream processing critically.

Concerning integrity metrics, Fig. 9a shows the number of unprocessed tuples. Since there are no failures in Case 1 and Case 2, this metric for both cases is zero. For Case 3, the average number of unprocessed tuples was around 530 thousand tuples, equivalent to 5.3% of the processing window. For Case 4, the average was around 1.2 million tuples, equal to 10.2% of the processing window.

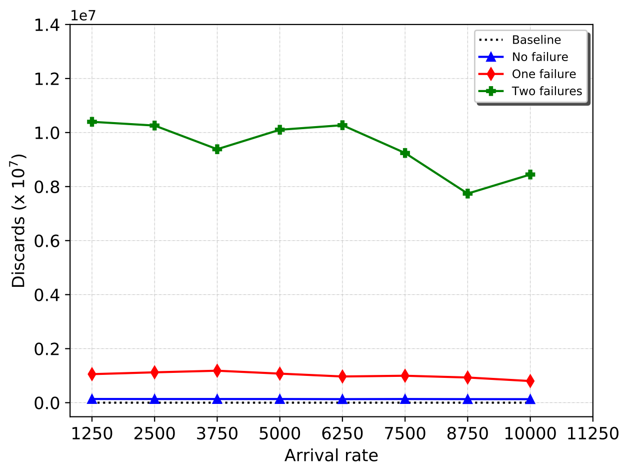
Using Eq. (4) to define an optimal checkpoint interval time is one approach to reduce this number. Less time between checkpoints implies a smaller period that an operator remains inactive and, consequently, it will process more tuples. However, frequent checkpoint increases overhead during failure-free executions (Casanova et al., 2015). Besides, an increase in the number of checkpoints directly affects tuple loss.

Another approach to alleviating this impact is to use the uncoordinated version of the Checkpoint Rollback-Recovery strategy. In this asynchronous approach, each node decides when to take its checkpoint independently (Goswami and Sahu, 2005), which avoids the requirement for blocking nodes. This procedure implies a reduced computational power during the node checkpoint, but there would be no information loss. However, the asynchronous checkpoint is risky due to the domino effect, when the recovery of a node depends on another node recovery (Gradvohl et al., 2014). Guermouche et al. (2011) present a solution for an uncoordinated checkpoint without a domino effect in applications that uses the Message Passing Interface (MPI) as its standard message exchange system between operators.

In turn, Fig. 9b presents the number of tuple losses. Due to the absence of fault-tolerance mechanisms in Case 1, there is no loss in this case. For Case 2, the average loss was 133 thousand tuples, equivalent to 1.33% of the processing window. The average loss for Case 3 was 1.1 million tuples, 10.1% of the processing window. Finally, for Case 4, the average



(a) Unprocessed.



(b) Loss.

Figure 9: Information loss and unprocessed tuples versus arrival rate.

loss was around 9.5 million tuples, equal to 95% of the processing window.

The following situation is one of the critical aspects of using CCRR in DaSP systems. In an emergency, the system could lose almost the same amount of tuples it processed. The system would lose one of two tuples. Due to the concept drift phenomenon, data streams are subject to changes in their probabilistic distribution that could occur in different types, such as gradual, sudden, incremental, or recurrent. For instance, sudden drifts appear abruptly and can completely change the data (Ramírez-Gallego et al., 2017). Discard all these tuples could result in losing a new or crucial change in the data probabilistic distribution that could lead to radically inaccurate decisions.

The combination of CCRR and Replication of Components (Gradvohl et al., 2014) could be a more reliable, long-term, and suitable approach to reduce these impacts. In this case, several operators running on different nodes would perform the same stream processing synchronously, in such a way that a failure in one node would not imply in unprocessed tuples. Then, on the checkpoint time, the system could recover the failed node as usual. This approach implies the increase in the computing power investment due to the necessity of at least duplicating operators, and the wasting of resources in failure-free executions, which could reduce the CCRR poor scalability (Casanova et al., 2015). Besides, it would also decrease stream loss since the system would not have to wait for a node recovery to resume stream processing.

As a final observation, researchers can use our model for the simulation of the CCRR strategy in DaSP systems. Results from the comparison with analytic models in Section 5 and the experiments in Section 6 demonstrated that the model is accurate to determine the performance and the impact the CCRR strategy has on DaSP systems. Also, since we built it in a user-friendly software such as ARENA, it enables the user's full control of the simulation, by changing different control parameters such as MTBF, arrival rate, recovery time, optimal checkpoint interval and whose operator will fail.

8 Conclusions

This paper presented a simulation model for evaluating the Coordinated Checkpoint-Rollback Recovery fault-tolerance strategy for Distributed Data Streams Processing Systems in both failure-free and failure-prone environments. With an error lower than 1.5% and 10.5% in these environments, respectively, we demonstrated that the simulation model is accurate to evaluate the proposed scenario. We also proposed two analytic models to predict information loss in failure-free and failure-prone environments, with an error lower than 1% and 11%, respectively.

Furthermore, we discussed how the CCRR negatively affects stream processing. We demonstrated through four case studies that using this strategy does not imply a severe impact in system performance in failure-free environments since the increase in mean waiting time, latency, and decrease in throughput was around 2%.

However, in emergencies, this strategy critically affects latency and throughput, and a high loss of information due to the system freezing during a global checkpoint. Therefore, we do not recommend using a pure coordinated checkpointing in the DaSP system. The use of process replication, in conjunction with this strategy or its asynchronous approach, with the attention to the domino effect, would be a more reliable approach to reduce both unprocessed and lost tuples.

Therefore, our work provides a reliable study on how much a coordinated checkpoint could affect the stream processing on a DaSP system, without the necessity to implement this strategy on a real architecture. Also, it provides an easy-to-use simulation model flexible enough to study different aspects of a DaSP environment, including fault-tolerance strategies.

Acknowledgments

The authors would like to acknowledge the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil – Finance Code 001.

References

Albertsson, L. (2006). Holistic debugging – enabling instruction set simulation for software quality assurance, *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pp. 96–103. https://doi.org/10.1109/MASCOTS.2006.26.

Böhm, S. and Engelmann, C. (2011). xSim: The extreme-scale simulator, *2011 International Conference on High Performance Computing Simulation*, pp. 280–286. https://doi.org/10.1109/HPCSim.2011.5999835.

Casanova, H., Robert, Y., Vivien, F. and Zaidouni, D. (2015). On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing, *Future Generation Computer Systems* **51**: 7–19. https://doi.org/10.1016/j.future.2015.04.003.

Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B. J. and Poulosky, P. (2016). Benchmarking streaming computation engines: Storm, flink and spark streaming, *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016* pp. 1789–1792. https://doi.org/10.1109/IPDPSW.2016.138.

Christine, G. and Emilie, G. (2005). Modelling of distributed system in one single simulation model: a way to study communications within distributed systems, *2005 IEEE Conference on Emerging Technologies and Factory Automation*, Vol. 1, pp. 7 pp.–703. https://doi.org/10.1109/ETFA.2005.1612594.

Daly, J. T. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps, *Future Generation Computer Systems* **22**(3): 303–312. https://doi.org/10.1016/j.future.2004.11.016.

de Assunção, M. D., Veith, A. S. and Buyya, R. (2018). Distributed data stream processing and edge computing: A survey on resource elasticity and future directions, *Journal of Network and Computer Applications* **103**(November 2017): 1–17. https://doi.org/10.1016/j.jnca.2017.12.001.

de Matteis, T. and Mencagli, G. (2017). Proactive elasticity and energy awareness in data stream processing, *Journal of Systems and Software* **127**: 302–319. https://doi.org/10.1016/j.jss.2016.08.037.

Ferreira, K., Stearley, J., Laros, III, J. H., Oldfield, R., Pedretti, K., Brightwell, R., Riesen, R., Bridges, P. G. and Arnold, D. (2011). Evaluating the viability of process replication reliability for exascale systems, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'11, ACM, New York, NY, USA, pp. 44:1–44:12. https://doi.org/10.1145/2063384.2063443.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. and Bouchachia, A. (2014). A survey on concept drift adaptation, *ACM Computing Surveys* **46**(4): 1–37. https://doi.org/10.1145/2523813.

Goswami, D. and Sahu, S. (2005). An efficient protocol for checkpoint-based failure recovery in distributed systems, *in* R. K. Ghosh and H. Mohanty (eds), *Distributed Computing and Internet Technology*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 135–144. https://doi.org/10.1007/978-3-540-30555-2_16.

Gradvohl, A. L. S. (2016). Investigating Metrics to Build a Benchmark Tool for Complex Event Processing Systems, *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, IEEE, Viena, pp. 143–147. https://doi.org/10.1109/W-FiCloud.2016.40.

Gradvohl, A. L. S. (2018). Metrics and tool for evaluating data stream processing systems, *2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 48–55. https://doi.org/10.1109/W-FiCloud.2018.00014.

Gradvohl, A. L. S., Senger, H., Arantes, L. and Sens, P. (2014). Comparing distributed online stream processing systems considering fault tolerance issues, *Journal of Emerging Technologies in Web Intelligence* **6**(2): 174–179. https://doi.org/10.4304/jetwi.6.2.174-179.

Guermouche, A., Ropars, T., Brunet, E., Snir, M. and Cappello, F. (2011). Uncoordinated checkpointing without domino effect for send-deterministic mpi applications, *2011 IEEE International Parallel Distributed Processing Symposium*, pp. 989–1000. https://doi.org/10.1109/IPDPS.2011.95.

Higashino, W. A., Capretz, M. A. and Bittencourt, L. F. (2016). CEPSim: Modelling and simulation of Complex Event Processing systems in cloud environments, *Future Generation Computer Systems* **65**: 122–139. https://doi.org/10.1016/j.future.2015.10.023.

Hoefler, T., Schneider, T. and Lumsdaine, A. (2010). LogGOPSim: Simulating large-scale applications in the loggops model, *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, ACM, New York, NY, USA, pp. 597–604. https://doi.org/10.1145/1851476.1851564.

Levy, S., Topp, B., Ferreira, K. B., Arnold, D., Hoefler, T. and Widener, P. (2014). Using simulation to evaluate the performance of resilience strategies at scale, *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, Springer International Publishing, Cham, pp. 91–114. https://doi.org/10.1007/978-3-319-10214-6_5.

Mehresh, R., Upadhyaya, S. J. and Kwiat, K. (2010). A multi-step simulation approach toward secure fault tolerant system evaluation, *2010 29th IEEE Symposium on Reliable Distributed Systems*, pp. 363–367. https://doi.org/10.1109/SRDS.2010.53.

Monnet, S., Morin, C. and Badrinath, R. (2004). Hybrid Checkpointing for Parallel Applications in Cluster Federations, *IEEE International Symposium on Cluster Computing and the Grid*, pp. 773–782. https://doi.org/10.1109/CCGrid.2004.1336712.

Moody, A., Bronevetsky, G., Mohror, K. and Supinski, B. R. (2010). Design, modeling, and evaluation of a scalable multi-level checkpointing system, *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11. https://doi.org/10.1109/SC.2010.18.

Mubarak, M., Carothers, C. D., Ross, R. and Carns, P. (2012). Modeling a million-node dragonfly network using massively parallel discrete-event simulation, *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 366–376. https://doi.org/10.1109/SC.Companion.2012.56.

Oldfield, R. A., Teller, P. J., Varela, M. R., Arunagiri, S., Seelam, S., Riesen, R. and Roth, P. C. (2007). Modeling the impact of checkpoints on next-generation systems, *Proceedings 24th IEEE Conference on Mass Storage Systems and Technologies, MSST 2007*, pp. 30–43. https://doi.org/10.1109/MSST.2007.4367962.

Park, A. J., Li, C., Nair, R., Ohba, N., Shvadron, U., Zaks, A. and Schenfeld, E. (2010). Flow: A stream processing system simulator, *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, pp. 1–9. https://doi.org/10.1109/PADS.2010.5471658.

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics, *Journal of Computational Physics* **117**: 1–19. https://doi.org/10.1006/jcph.1995.1039.

Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M. and Herrera, F. (2017). A survey on data preprocessing for data stream mining: Current status and future directions, *Neurocomputing* **239**: 39–57. https://doi.org/10.1016/j.neucom.2017.01.078.

Shchur, L. N. and Shchur, L. (2015). Parallel discrete event simulation as a paradigm for large scale modeling experiments, *Selected Papers of the XVII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2015), Obninsk, Russia, October 13-16, 2015.*, pp. 107–113. Available at http://ceur-ws.org/Vol-1536/paper16.pdf.

Tikotekar, A., Vallee, G., Naughton, T., Scott, S. L. and Leangsuksun, C. (2007). Evaluation of fault-tolerant policies using simulation, *2007 IEEE International Conference on Cluster Computing*, pp. 303–311. https://doi.org/10.1109/CLUSTR.2007.4629244.

Zheng, G., Kakulapati, G. and Kale, L. V. (2004). Bigsim: a parallel simulator for performance prediction of extremely large parallel machines, *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pp. 78–. https://doi.org/10.1109/IPDPS.2004.1303013.

Zheng, G., Wilmarth, T., Jagadishprasad, P. and Kalé, L. V. (2005). Simulation-based performance prediction for large parallel machines, *International Journal of Parallel Programming* **33**(2-3): 183–207. https://doi.org/10.1007/s10766-005-3582-6.