Adaptive Synchronization and Pacing Control for Visual Interactive Simulation

ZHUOXIAO MENG, Technical University of Munich, Germany and Huawei Munich Research Center, Germany
MINGYUE GAO, Huawei Munich Research Center, Germany
MARGHERITA GROSSI, Huawei Munich Research Center, Germany
ANIBAL SIGUENZA-TORRES, Technical University of Munich, Germany and Huawei Munich Research Center, Germany
STEFANO BORTOLI, Huawei Munich Research Center, Germany
CHRISTOPH SOMMER, TU Dresden, Germany
ALOIS KNOLL, Technical University of Munich, Germany

Parallel and distributed computing enable the execution of large and complex simulations. Yet, the usual separation of (headless) simulation execution and (subsequent, offline) output analysis often renders the simulation endeavor long and inefficient. Recently, Visual Interactive Simulation (VIS) tools and methods that address this end-to-end efficiency are gaining relevance, offering *in-situ* visualization, real-time debugging, and computational steering. Here, the typically distributed computing nature of the simulation execution poses synchronization challenges between the headless simulation engine and the user-facing frontend required for Visual Interactive Simulation. To the best of our knowledge, state-of-the-art synchronization approaches fall short due to their rigidity and inability to adapt to real-time user-centric changes. This paper introduces a novel adaptive algorithm to dynamically adjust the simulation's pacing through a buffer-based framework, informed by predictive workload analysis. Our extensive experimental evaluation across diverse synthetic scenarios illustrates our method's effectiveness in enhancing runtime efficiency and synchronicity, significantly reducing end-to-end time while minimizing user interaction delays, thereby addressing key limitations of existing synchronization strategies.

CCS Concepts: • **Computing methodologies → Modeling and simulation**; **Interactive simulation**; **Real-time simulation**; *Simulation tools.*

Additional Key Words and Phrases: Visual interactive simulation, In-situ visualization, Human-in-the-loop simulation, Adaptive simulation synchronization

ACM Reference Format:
Zhuoxiao Meng, Mingyue Gao, Margherita Grossi, Anibal Siguenza-Torres, Stefano Bortoli, Christoph Sommer, and Alois Knoll. 2024. Adaptive Synchronization and Pacing Control for Visual Interactive Simulation. 1, 1 (June 2024), 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Zhuoxiao Meng, Technical University of Munich, Department of Informatics, Munich, Germany and Huawei Munich Research Center, Intelligent Cloud Technologies Laboratory, Munich, Germany; Mingyue Gao, Huawei Munich Research Center, Intelligent Cloud Technologies Laboratory, Munich, Germany; Margherita Grossi, Huawei Munich Research Center, Intelligent Cloud Technologies Laboratory, Munich, Germany; Anibal Siguenza-Torres, Technical University of Munich, Department of Informatics, Munich, Germany and Huawei Munich Research Center, Intelligent Cloud Technologies Laboratory, Munich, Germany; Stefano Bortoli, Huawei Munich Research Center, Intelligent Cloud Technologies Laboratory, Munich, Germany, {firstname.lastname}@huawei.com; Christoph Sommer, TU Dresden, Faculty of Computer Science, Dresden, Germany, cms-labs.org/people/sommer; Alois Knoll, Technical University of Munich, Department of Informatics, Munich, Germany, k@tum.de.

1 INTRODUCTION

Analysis and visualization of simulation data are typically performed after the simulation is completed (post-hoc), without human interaction during the execution (e.g., [31]). To minimize the number of simulations, users often configure the simulation to output as much data as possible, which can be a lengthy and resource-intensive process, especially in a cloud environment [35].

Visual Interactive Simulation (VIS) [26] offers a promising solution to this problem. It can generate a dynamic, real-time display of simulations, allowing user interaction and control as the simulation progresses. Unlike the traditional method, which requires extensive data storage, users can customize the simulation output on-the-fly, for example by filtering or implementing user-defined functions. This process, known as *in-situ* visualization [17, 20], facilitates rapid and free-form exploration of the simulated domain. In addition, VIS allows users to promptly alter the dynamics of the running simulation, such as changing parameters and conducting "what-if" explorations, without having to restart the simulation from scratch [25]. Recent research has demonstrated the effectiveness of carefully chosen visualization designs and interaction techniques in allowing users to explore urban transportation [33], fluid dynamics [16], and networked systems [21], among others.

Maintaining a consistently updated representation of the simulation state in the visualization application is of utmost importance. This ensures an appropriate user experience, preventing user actions from being based on outdated simulation states. At the same time, it is important to control the overhead associated with synchronization in order to maintain the best possible system performance [28]. Striking an adaptive balance between user experience and overhead is therefore an important concern.

At present, synchronization in VIS adheres to a conservative step-based methodology, following either a sequential or a parallel model [7]. In the sequential model, the simulation and visualization processes alternate on the operational timeline [2] and run one after the other. While this ensures precise synchronization, it does so at the expense of system efficiency [20]. Conversely, the parallel model offers improved system performance by allowing simulation and visualization processing to occur simultaneously. However, it introduces the challenge of maintaining the synchronicity. A common approach, as detailed in Section 3.2, is to synchronize the simulation and visualization at regular time intervals. However, the longer the interval, the greater the risk of desynchronization between the two processes, resulting in a poor Quality of Experience (QoE) for the user.

This paper presents an adaptive synchronization strategy that utilizes runtime data to adjust the simulation's pacing to the visualization task workload. The implementation is supported by a novel buffer-based framework that enables dynamic adjustment of synchronization points through buffer management. Experimental evaluation using synthetic yet generic test cases shows that the proposed approach achieves optimal system performance and improves interaction efficiency compared to the current state-of-the-art. Our approach is also characterized by its self-adaptive nature, which requires less prior knowledge about the simulated scenarios, making it valuable in diverse situations. In addition, the proposed framework is notable for its ease of implementation, extension, and backward compatibility with the state-of-the-art methods, allowing for seamless integration into existing VIS setups.

The structure of this paper is as follows: Section 2 provides the background and motivation for our research. Section 3 examines related work, with an emphasis on the conventional method referred to as rigid synchronization. In Section 4, we present a formulation for evaluating synchronization methods in VIS systems. Section 5 presents our novel synchronization approach, explaining the architecture and strategy in detail. Section 6 presents a comprehensive evaluation of our approach using various synthetic scenarios. Finally, in Section 7, we summarize the contributions of our study, discuss its limitations, and suggest future research directions.

2 BACKGROUND AND MOTIVATION

Our research initiative is rooted in the development and application of a large-scale vehicle traffic simulation service platform. This platform hosts a city-scale timestepped microscopic traffic simulator in the backend, coupled with a client application, hereafter referred to as the Visualizer. This Visualizer is designed to provide support for real-time simulation data analysis and visualization, establishing the platform as an efficient and intuitive tool for city-scale urban traffic management studies.

Targeting metropolis-scale, the simulation models involve millions of agents such as vehicles and pedestrians. A naive approach of transferring and rendering the full simulation state's updates is computationally burdensome, resource intensive, and yields unsatisfactory performance. Therefore, in order to support real-time simulation data visualization and analysis, an interactive data reduction strategy is employed, allowing users to selectively investigate simulation data based on its contextual relevance. For example, one of the main features of the Visualizer is to allow users to change the Field of View (FoV) by zooming, panning, and focusing. By default, only agents within the FoV are considered relevant and will be displayed, greatly reducing the amount of data transferred and processed. In addition, users can customize the resolution and level of detail in the visualization to meet their specific needs, displaying different levels of aggregated analytics relevant for the specific FoV. For example, when visualizing the whole city or a very large district, only macroscopic traffic metrics would be relevant to provide the required overview of the simulated city. Thus, users can choose to display coarser and lower resolution analytics, which can further reduce the workload on the Visualizer.

In addition, the Visualizer allows users to interactively define and select relevant metrics for computation, and to adjust simulation parameters at runtime, such as performing "what-if" experiments using the well-known simulation cloning paradigm [13]. Therefore, the capability of supporting an efficient interaction of users with the large-scale headless simulation running in the background is pivotal in this platform, making synchronization between the engine and the Visualizer a key factor in the usability and user QoE. The high computational demands on both the Simulator and the Visualizer suggest that the sequential synchronization (see Section 1) would prolong runtimes, and a parallel operation of the two is preferable. However, a parallel approach can introduce interaction delays in the VIS, as detailed in Section 4. For instance, when users alter the range of the spatial filter by switching their FoV, newly selected agents will not appear immediately due to the simulation time lag between the user's view time and the actual simulation time. While this delay is acceptable if it does not impede decision making, it should be minimized to improve the user experience.

3 RELATED WORK

3.1 Tightly and Loosely Coupled VIS

Since the 1990s, the field of computational steering, which integrates analysis and visualization into simulation workflows, has significantly evolved [14]. At this early stage, particularly due to the small scales of simulated scenarios, the dominant method was the **Tightly Coupled** approach [17], also termed as the **Synchronous** approach in other literature [2, 15]. In this method, the visualization code is embedded in the simulation code and only one process is allowed to use computational resources at any given time, thereby enforcing a step-by-step execution of simulation and visualization. Naturally, this leads to the implementation of sequential synchronization. Despite its inefficiency in terms of end-to-end performance, the simplicity [17] of this approach leads to its widespread adoption in various open-source frameworks such as SCIRun [27], Libsim [30], Paraview Catalyst [1], and commercial solutions like AWS SimSpace Weaver[1], which remain in use today.

The 2000s witnessed the introduction of advances in *in-situ* visualization and steering for large-scale simulations on supercomputers [24, 32]. During this time, the **Loosely Coupled** approach, also known as the **Asynchronous** approach, emerged as a response to the high computational demands of both visualization and simulation [15, 17]. The loosely coupled approach involves the allocation of dedicated computational resources to visualization and simulation independently. This enables them to run simultaneously, providing improved flexibility and scalability compared to the tightly coupled approach. However, it also introduces the requirement for tight coordination of workflows between visualization and simulation, which requires careful synchronization, as explained in [15].

3.2 Rigid Synchronization

In loosely coupled VIS systems, a prevalent synchronization strategy involves aligning the simulation and visualization processes at regular intervals, a method we refer to as **Rigid Synchronization** in this paper. As illustrated in Fig. 1, this approach involves defining, at the start of the simulation and throughout the entire duration of the simulation, a synchronization interval duration T_{rigid}, representing the simulation time span between synchronization points. At each synchronization point, the simulation sends simulation data from its most recent interval to the Visualizer, while the Visualizer forwards commands related to the user interactions to the simulation. After each synchronization point, the simulation and visualization proceed simultaneously, each for a fixed period of T_{rigid}, and then wait until both have finished in order to proceed with the next. Notably, this results in the simulation time always being at least T_{rigid} ahead of the visualization.

Examples of this approach include the use of MPI barriers for synchronization in turbulent transition simulations by Buffat et al. [6] and the definition of "output-steps" (which serves as their synchronization interval) in the interactive simulation rendering framework proposed by Kawamura et al. [16]. However, in these studies, visualization components and data analysis tasks are configured before the simulation begins. User interaction is limited to only modifying simulation parameters, which does not significantly change the visualization workload. Thus, the workload of the entire system is typically predictable before the execution.

Our scenario significantly differs due to its dynamic nature, where visualization demands can shift in real-time in response to user interactions, presenting challenges in applying rigid synchronization effectively. First, a suitable T_{rigid} is difficult to set beforehand. Correct choice of T_{rigid} is strongly affected by the particular characteristics of the system's workload. Setting it too short may not provide sufficient asynchronicity and negatively impact the runtime performance due to frequent synchronization, whereas setting it too long could result in significant temporal disparities between the visualization and simulation processes. Second, the fixed nature of the synchronization interval does not accommodate the fluctuating demands of user interactions. An interval that is optimal in one phase may become less so in another one, as the visualization tasks can change due to a user's real-time visualization requirements. Therefore, we need a more flexible approach than what rigid synchronization with a one-size-fits-all strategy can provide.

[1]https://aws.amazon.com/simspaceweaver/ Retrieved: 18.04.2024

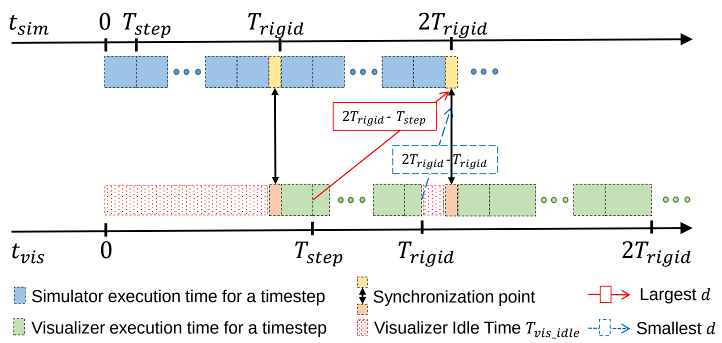


Fig. 1. Timelines for the simulation timestamp (t_{sim}), the visualization timestamp (t_{vis}), and the largest, smallest interaction delay in a rigid synchronization approach.

3.3 Other Synchronization approaches

An alternative asynchronous strategy involves skipping over older simulation data to display only the most recent updates, ensuring the visualization remains synchronized with the ongoing simulation. This allows the simulation to proceed uninterrupted. While this method can achieve the optimal runtime performance, it may lead to the omission of important intermediate updates. A notable example is the *in-situ* visualization pipeline presented by Krüger et al. [18] in 2022 for neuronal network simulations. Another example can be found in [3]. However, the effectiveness of this approach, like rigid synchronization, relies on both the visualization and simulation components having predictable and consistent workloads. For example, Krüger et al. [18] argued that in their specific scenario, the processing of visualization at each time could generally be completed before the data exchange deadline with the simulation, resulting in minimal instances of data loss. However, they also acknowledged that this is a tentative assumption that needs to be further explored in future work. Particularly in scenarios like ours, where the demands on visualization can vary significantly, this strategy might result in an unmanageable temporal gap between simulation and visualization, potentially leading to uncontrollable data loss.

In the paper introducing DAR-CI [23], the API used in our platform for traffic simulation control, a discrete event-based synchronization method is proposed. This method allows simulation and connected client applications to run asynchronously, with client commands mapped to external events in the simulation's update logic. However, to prevent delays in the execution of user commands, users are required to send a pause event at a predetermined simulation time. This approach is not feasible for our use case due to the unpredictable timing of user commands, as we cannot anticipate when or what type of command users might issue.

We have also explored alternative simulation frameworks, including Dynamic Data Driven Applications Systems (DDDAS) [8] and Digital Twin (DT) systems [5], which inherently require simulations to exchange information with external data sources, thus making synchronization a necessary component to consider. The majority of these applications applied a synchronization approach prevalent in the co-simulation domain [12] characterized by conservative and optimistic synchronization with rollback capabilities [11]. Innovations like [4], designed to save synchronization energy for distributed DDDAS, are not relevant for our targeted VIS use case.

3.4 Adaptive Approach in VIS

The application of adaptive techniques in simulations involving online data processing and visualization has also been explored. However, these studies focus primarily on aspects such as data reduction [9, 19, 36] and resource allocation [10, 20, 29], with the main goal of improving overall performance, i.e., minimizing end-to-end time. Interactive elements are typically not integrated or required in these approaches. While these studies have different goals than ours, they demonstrate the feasibility of predicting runtime conditions and adapting accordingly in VIS systems.

4 PROBLEM FORMULATION

The objective of this work is to propose a novel synchronization approach that balances runtime efficiency with interaction timeliness, thereby enhancing user QoE. We measure this balance using two conflicting metrics:

(1) **Visualizer Idle Time ($T_{\text{vis\_idle}}$)**: This measures the duration (**wall-clock time**) the Visualizer remains idle while waiting for new simulation data (an example is illustrated in Fig. 1). Lowering $T_{\text{vis\_idle}}$ is vital for uninterrupted visual display and overall system performance. However, lowering it to zero may not always be feasible given the limitation of simulation performance. It is important to clarify that enhancing the performance of the simulation itself is not within the focus of this paper.

(2) **Average Delay of Interaction ($\bar{d}$)**: This represents the synchronicity between the Visualizer and the simulation. We identify each interaction with an index i. The timestamp the interaction is triggered on the Visualizer side is denoted as t_{vis_i}, and the simulation timestamp it applies to is denoted as t_{sim_i}. The interaction delay is thus quantified as $d_i = t_{\text{sim}_i} - t_{\text{vis}_i}$. We denote the average value of all interactions' delay as $\bar{d}$. Minimizing $\bar{d}$ ensures timely and relevant user decisions.

We thus define our challenge as a Multi-Objective Optimization (MOO):

$$\min_{x \in X} \left(T_{\text{vis\_idle}}\left(x\right), \bar{d}\left(x\right) \right) \text{ subject to: } d_i \leq d^{\max} \tag{1}$$

In this context, X represents the collection of possible synchronization techniques. Every x is an instance of a distinct synchronization approach (e.g, Rigid Synchronization) accompanied by its unique set of parameters (e.g., T_{rigid}). To ensure that critical decision-making interactions are not delayed, a constraint called the *maximal acceptable interaction delay* is introduced, denoted as $d^{\max}$. This constraint ensures that the delay of any interaction during a run does not exceed a certain threshold.

Reducing both the visualizer's idle time $T_{\text{vis\_idle}}$ and the average delay $\bar{d}$ at the same time poses a significant challenge, given that they typically exhibit an inverse correlation. In other words, a decrease in one tends to result in an increase in the other, and vice versa. The two extremes of the inverse relationship are exemplified by sequential synchronization and post-hoc data processing. The former approach, sequential synchronization, reduces the delay to zero, i.e., $d_i = 0$, implying an immediate user interaction. However, this method leads to the maximum $T_{\text{vis\_idle}}$, which could be equivalent to the entire simulation execution time (excluding the data transfer time), due to the lack of parallel processing. On the other hand, post-hoc data processing represents the other extreme, where $T_{\text{vis\_idle}}$ is virtually non-existent since all data is instantly available. Nonetheless, in this scenario, the interaction delays are deemed to be infinitely large as the simulation has already been completed.

Regarding Rigid Synchronization (Section 3.2), adjusting the input parameter T_{rigid} is typically how users balance these two metrics. The delay of each interaction is fixed with a predetermined synchronization interval T_{rigid}, and the larger T_{rigid}, the greater the $\bar{d}$. As shown in Fig. 1, the largest interaction delay occurs when an interaction is made after the first simulation timestep of a synchronization interval, which can be calculated as $2T_{\text{rigid}} - T_{\text{step}}$. Here, T_{step} represents the duration of a simulation timestep. On the other hand, the smallest interaction delay is encountered when an interaction is made at the end of a synchronization interval, which is equal to T_{rigid}. Assuming that an interaction occurs at each timestep, the average delay $\bar{d}$ is calculated as $\frac{3T_{\text{rigid}} - T_{\text{step}}}{2}$. With respect to $T_{\text{vis\_idle}}$, as depicted in Fig. 1, it embodies the aggregate of all idle durations within each synchronization interval, and it is dictated by the workload balance between the Visualizer and the simulation within each T_{rigid}. Its exact value cannot be calculated without knowledge of the specific simulated scenario being considered. However, in general, a lengthier T_{rigid} typically suggests a greater level of parallelism, which results in a more balanced distribution of workload and consequently, a reduced $T_{\text{vis\_idle}}$. The experiment carried out in this research corroborated this observation as well (Section 6.4).

In our suggested buffer-based synchronization method (refer to Section 5 for more details), the idle time of the Visualizer is a result of its capability to process simulation data faster than the simulation produces it. Our approach to mitigate this, without accelerating the simulation, is to let the simulation run further during its faster stage, i.e., generating simulation data for later use while the Visualizer, for example, is performing some heavy tasks. The more we allow the simulation to run in advance, the more data can be buffered for the Visualizer, subsequently reducing the $T_{\text{vis\_idle}}$. However, this also leads to a larger time difference between the frontend and the backend, resulting in prolonged user interaction delays ($\bar{d}$), and vice versa. Thus, the extent to which the simulation is allowed to advance, an input parameter termed as *buffer capacity*, can be adjusted by users to fine-tune the balance between $T_{\text{vis\_idle}}$ and $\bar{d}$ according to their preference.

From our perspective, addressing this MOO (Eq. 1) issue directly, especially pinpointing the ideal synchronization and its optimal parameters using an analytical model without running the system, is not practical. Because the true outcomes of the metrics, specifically $T_{\text{vis\_idle}}$ and $\bar{d}$, are significantly affected by the execution performance at each simulation time of the simulation and visualization. Developing a model to calculate these is overly challenging due to the excessive correlated input factors, including the details of the current scenario, the variable workload throughout the process, and the hardware specifications such as CPU and network capabilities. Furthermore, it is even more impractical and error-prone to formulate a mathematical model for user interactions considering their random nature. Hence, this paper suggest an auto-adapt strategy, i.e., a heuristic approach, that monitors the system during runtime and consistently fine-tunes the synchronization parameter accordingly to meet the objectives.

5 ADAPTIVE SYNCHRONIZATION AND SIMULATION PACING CONTROL

In this work we target to better support interactive usage of simulation for exploratory endeavours, with the further goal of addressing the problem at scale, while keeping the system as generally reusable and extensible as possible. Hence, the most fitting option is to design a VIS framework that loosely couples simulation engine and the Visualizer. Fig. 2 illustrates the overall design of the proposed architecture. The **Simulator** and the **Visualizer** are presented as separate logical components. The coordination between the Visualizer and the Simulator is handled by an intermediary component called **Controller**. Within the Controller, the **Interaction Collector** and **Simulation Executor** are responsible for receiving interaction control commands from users and the step-based execution of the simulation engine respectively. The data exchange and synchronization are mediated by a dynamically adjustable **Bounded Buffer**. Namely, the simulation can be triggered by the Simulation Executor only if the Bounded Buffer has space to receive the corresponding data from the simulation engine. The **Pacing Controller** utilizes therefore real-time insights related to simulation and interaction demands to adjust the capacity of the Bounded Buffer based on specific policy, thereby

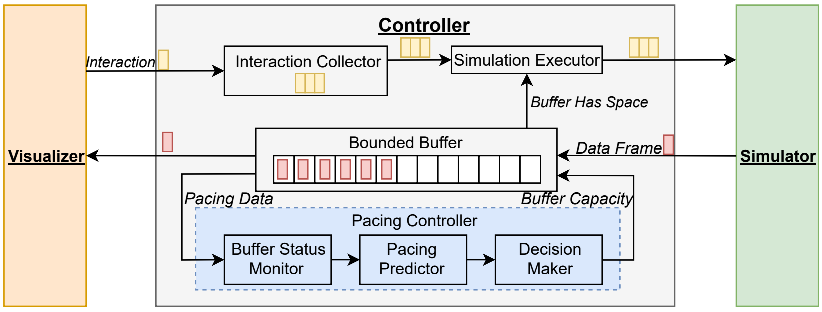


Fig. 2. Overview of the proposed architecture.

Algorithm 1: Pseudocode for Simulator Workflow

Input : updateInterval Δt

Initialization : currentTime $t_{sim} \leftarrow t_{\text{init}}$;
interactions $I \leftarrow None$

while *simulation is running* **do**
- $I \leftarrow$ wait_for_signal();
- apply_interactions(I);
- $t_{sim} \leftarrow t_{sim} + \Delta t$;
- $df \leftarrow$ simulation_run_until(t_{sim});
- reply_with(df);

end

Algorithm 2: Pseudocode for Visualizer Workflow

Input : updateInterval Δt

Initialization : currentTime $t_{vis} \leftarrow t_{\text{init}}$;
interactions $I \leftarrow None$

while *simulation is running* **do**
- $df \leftarrow$ request_dataframe(I);
- $t_{vis} \leftarrow t_{vis} + \Delta t$;
- $I \leftarrow$render_process(df, t_{vis}) ;

end

controlling also the simulation pacing. In this section, we will provide a detailed explanation of the functionality of each component, as well as the adaptive pacing policy proposed in this study.

5.1 Simulator and Visualizer Workflow

In our framework, both the Simulator and the Visualizer are involved in a repetitive, cyclic process. Each cycle involves advancing the simulation time by a fixed interval, denoted as Δt, which is predetermined and by default equal to one simulation step, i.e., T_{step}.

The Simulator workflow execution loop, as outlined in Algorithm 1, starts with waiting for a trigger signal. Once the signal is received, the simulation executes the interaction actions based on the incoming request and continues for a duration of interval Δt. Upon completion, a data unit called **Data Frame** (*df*) including the desired simulation output is returned.

The Visualizer also operates based on synchronous blocking calls, as shown in Algorithm 2. Within each loop, it sends out a message with the latest user interaction control commands I from the last period and receives a *df* in response. Notably, the interaction control message can be empty. The Visualizer can then processes the received *df* to update its display and capture new user interactions. Both the Simulator and the Visualizer repeat their workflow until the simulation concludes or the user terminates the process.

The design goal here is to keep the system as unobtrusive as possible and reduce the need for altering the Simulator and Visualizer codebases. Direct coupling of the Visualizer and Simulator following this workflow would lead to the rigid synchronization approach (Section 3.2), with Δt serving as the synchronization interval T_{rigid}. One of the distinguishing features of this work is the introduction of the Controller design that serves as a middleware, linking the Visualizer with the Simulator. This component is designed to channel user interactions (I) from the Visualizer to the Simulator and vice-versa for simulation data (*df*), while handling the synchronization between the two.

5.2 Data Exchange and Synchronization Mechanism

In this section we explain how the Controller interacts with the Simulator and the Visualizer. The Simulator progresses by Δt upon receiving a trigger message that includes user interaction commands and returns a *df* as the response. As depicted in Fig. 2, these trigger messages are initiated by the Simulation Executor within the Controller, with each message including all the interactions accumulated by the Interaction Collector. The Bounded Buffer is implemented as a thread-safe queue. As long as there is available space in the buffer, the Simulation Executor can keep sending trigger messages and enqueue the received *df* in the queue. Maintaining the sequence of *df* sent from the Simulator to the Controller is essential for the implementation, yet it is not challenging to achieve. For example, TCP and several open-source libraries such as gRPC[2] (which we used) and ZeroMQ[3] can offer this guarantee.

On the other hand, during each interaction step Δt, the Visualizer sends an interaction message to the Controller and receives the oldest *df* dequeued from the Bounded Buffer (i.e., with FIFO order). During each cycle, both the Simulator and the Visualizer independently progress their computation for an identical duration (Δt) and execute a single enqueue-dequeue process. Thus, the timestamp of the dequeued *df* consistently matches the timestamp of the Visualizer's request.

The number of *df* accumulated in the buffer, denoted as n_{df}, reflects the time lag of the Visualizer behind the Simulator. The delay of an interaction can thus be quantified as $n_{\text{df}} \times \Delta t$, where n_{df} is the buffer size when the interaction request reaches the Controller. When the buffer becomes full, signaling that the simulation is significantly ahead of the Visualizer, the Controller will temporarily halt the initiation of new simulation updates while the visualizer continues its processing. It becomes therefore intuitive that the capacity of the Bounded Buffer determines the *maximal acceptable interaction delay* ($d^{\max}$) in the system. Storing extra *df*(s) in the buffer offers the benefit of averting potential future data shortages. Should there come a time when the Visualizer's rate of consuming *df* surpasses the rate at which the simulation generates it, the *df* already in storage can be sent back to the Visualizer immediately upon request, effectively eliminating any waiting periods.

[2]https://grpc.io/ Retrieved: 18.04.2024
[3]https://zeromq.org/ Retrieved: 18.04.2024

Unlike rigid synchronization, which strictly synchronizes and transfers data at predetermined, coarse-grained time intervals, the buffer-based approach adds flexibility by adjusting the delay to some extent based on the actual system load. The interaction delay occurs and increases only when the buffer starts to fill and grow, i.e., when the simulation actually runs faster than the visualization. As the simulation slows down, the n_{df} decreases, resulting in a reduction of the temporal gap and a more timely interaction. After the buffer is emptied, synchronization can occur at each Δt, i.e., the finest synchronization unit, with minimal interaction delay and no negative impact on runtime performance.

However, if the Visualizer is constantly running slower than the simulation, the buffer will always be full. This persistent state results in sustained maximum delays, but without any improvement in runtime efficiency, because the overall system performance is still limited by the slow processing speed of the Visualizer. In this paper, this issue is defined as the **Overbuffering** problem. To address it, in the following section, we present the Pacing Controller, which is a component that can adaptively modify the buffer capacity during runtime according to the real-time workload.

5.3 Simulation Pacing Control

The Pacing Controller is responsible for modifying the capacity of the Bounded Buffer in order to regulate the pacing of the simulation process. The underlying principle is to increase the buffer capacity (up to the maximum limit specified by $d^{\max}$) when the visualization is expected to be faster than the simulation in the near future. This allows for more simulation data to be stored in the buffer for later use, reducing the idle time the Visualizer will experience. However, if the visualization is expected to be slower, or if the existing df(s) in the buffer are deemed sufficient for the near future, the buffer capacity can be kept relatively small. This would slow down the simulation to match the pace of the Visualizer, improving system responsiveness without sacrificing overall performance. Essentially, the overall system approaches the maximal overall performance without affecting QoE.

As illustrated in Fig. 2, the Pacing Controller consists of three modules: the **Buffer Status Monitor**, the **Pacing Predictor**, and the **Decision Maker**. The Buffer Status Monitor tracks runtime data such as the rate at which the Simulator enqueues df(s) and Visualizer dequeues them, along with their respective counts. The Pacing Predictor is the component to make heuristic predictions about the processing speed of the Simulator and Visualizer. Various time series data prediction models, such as Exponential Smoothing, SARIMA, and LSTM can be applied for this purpose [22].

Notably, the arbitrary nature of user interactions poses a significant challenge for prediction accuracy of the Visualizer's workload. However, there would still exist a certain degree of consistency of workload patterns following each interaction or group of interactions. As long as the interactions do not vary excessively and with very high frequency, which is uncommon in large-scale simulations, this consistency allows for sufficient predictability. Additionally, the Pacing Predictor could potentially utilize the semantics of the interactions as supplementary information to enhance its predictive accuracy. It is important to clarify that the specific methodologies for prediction are not the primary focus of this work. Rather, our contribution lies in the conceptualization and implementation of the Pacing Predictor as a component within the overall pacing control system.

After receiving the pacing predictions for both the Simulator and the Visualizer from the Pacing Predictor, the Decision Maker is in a position to determine the buffer's capacity. This dynamically determined capacity is key in allowing the Simulator to match the pace of the Visualizer at runtime, overcoming the limitations of predefining a fixed value for the entire run.

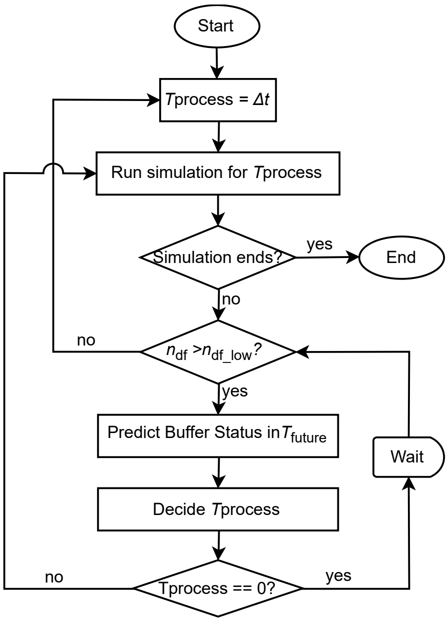


Fig. 3. Adaptive pacing policy workflow.

5.4 Adaptive Pacing Policy

In this section, we delve into the details of the adaptive pacing policy that is proposed in this study. The policy includes a predefined input parameter called the *desired buffer lower bound*, denoted as $n_{\text{df\_low}}$. This parameter serves as a safeguard, indicating the minimum quantity of *df* that ought to be maintained in the Bounded Buffer to avert data shortages. The main objective of this policy is to control the simulation pacing in order to strike a balance between two opposing objectives concerning the buffer states. First, it aims to keep the number of *df* in the Bounded Buffer remains above the predefined $n_{\text{df\_low}}$. Second, it also aims to minimize the number of *df* in the Bounded Buffer, thereby reducing the time lag between the Visualizer and the Simulator. The policy operates by constantly estimating the future workloads for both the simulation and visualization process and adjusts the execution of the simulation based on these predictions. The outcome of each decision is referred to as the *simulation processing time*, denoted by T_{process}. This value determines if the simulation should keep running, for how long, or if it should be paused.

The comprehensive flow chart of the proposed policy is illustrated in Figure 3. As shown, the simulation initially sets T_{process} to Δt, continuing until the current buffer size, i.e., n_{df}, surpasses the predefined $n_{\text{df\_low}}$. The Pacing Predictor is then employed to estimate the upcoming workload, specifically the processing time required for each Δt by both the Simulator and the Visualizer, for an upcoming period referred to as T_{future}. These estimations are utilized to determine the future states of the buffer, namely, the predicted buffer size at any future moment, denoted as $n_{\text{df\_pre}}(t)$, with t ranging from 0 (i.e., the present) to T_{future}. An example is illustrated in Fig. 4, which also highlights the significance of taking into account the user-defined maximum buffer capacity ($d^{\max}/\Delta t$) in these future buffer state calculations.

Based on the predictions of the buffer state, the policy determines the outcome value T_{process}. Specifically, the chosen value should mark the transition point when the predicted buffer size goes from being below $n_{\text{df\_low}}$ to consistently

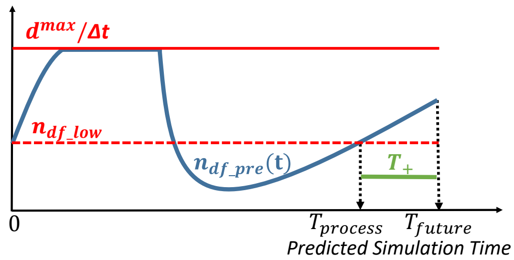


Fig. 4. Illustration of a predicted buffer status in T_{future} and the decision of T_{process}

exceeding it, as shown in Fig. 4. We denote T_+ as the set of subsequent timesteps preceding T_{future}, at which the predicted buffer size surpasses the $n_{\text{df\_low}}$:

$$T_+ := \{t_+ \in [0, T_{\text{future}}] \mid n_{\text{df\_pre}}(t) \geq n_{\text{df\_low}}, \forall t \in [t_+, T_{\text{future}}]\} \tag{2}$$

and then the chosen value of T_{process} is:

$$T_{\text{process}} = \begin{cases} min\{T_+\}, & \text{if } T_+ \neq \emptyset \\ T_{\text{future}}, & \text{otherwise} \end{cases} \tag{3}$$

This suggests that before reaching this critical point (T_{process}), there is a possibility that the buffer will hold less data than $n_{\text{df\_low}}$. Therefore, it is crucial to initiate the simulation without delay to prevent the upcoming buffer size from shrinking more. However, from this point (T_{process}) until the end of the forecast period (T_{future}), the buffer storage is expected to remain sufficient. Thus, executing the simulation for the period of T_{process} ought to be adequate. Once the simulation has been run for this length of time, a new prediction should be made to reassess the situation, avoiding excessive data accumulation affecting the interaction delay. In terms of implementation, this is achieved by increasing the capacity of the Bounded Buffer to its maximum limit ($d^{\text{max}}/\Delta t$) until the simulation is continuously triggered for $T_{\text{process}}/\Delta t$ times.

It's worth noting two particular circumstances of interest. The first involves a condition where the predicted buffer size is consistently above $n_{\text{df\_low}}$ throughout T_{future}, i.e., $T_+ = [0, T_{\text{future}}]$ and $T_{\text{process}} = 0$. This implies that the current buffer already has sufficient data for the forthcoming period, possibly due to a gradual slowdown in the Visualizer over time. Consequently, the simulation is commanded to wait, as is shown in Fig. 3. During this waiting time, predictions are continuously updated as the Visualizer processes the cached df, until either a T_{process} is determined or current buffer size is below the $n_{\text{df\_low}}$. The second scenario occurs when the predicted buffer size at the end of T_{future} is below $n_{\text{df\_low}}$, that is, $n_{\text{df\_pre}}(T_{\text{future}}) < n_{\text{df\_low}}$ and consequently $T_+ = \emptyset$, indicating a potential data shortage in the entire T_{future}. The simulation is then allowed to continue at full speed throughout T_{future}, i.e., $T_{\text{process}} = T_{\text{future}}$.

By periodically assessing and predicting the buffer's state, this proposed policy aims to maintain an optimal balance: minimizing the Visualizer idle time by avoiding an empty buffer, while also limiting the accumulation of excessive df, thereby ensuring efficient interaction while optimal runtime performance and accounting for the demands of real-time visualization tasks. The choice of $n_{\text{df\_low}}$ could intuitively reflect the user's prioritization. A large value for $n_{\text{df\_low}}$ allows the Bounded Buffer to store more df most of the time, thus reducing the risk of data shortages. However, this also means that there is an increase in interaction delays. On the other hand, a small value means a greater concern of users for reducing interaction delays, but it also increases the likelihood of encountering data shortages more frequently. As for the T_{future}, it can also be an adaptive value. In our trials, we experimented with setting T_{future} relevant to the time interval between occurrences when the simulation needs to wait, yielding to satisfactory outcomes. Additionally, factors such as prediction accuracy could also be used to further fine-tune T_{future} and $n_{\text{df\_low}}$ at runtime, although we did not delve into such specifics.

6 EXPERIMENTAL EVALUATION

The experiments are carried out on the traffic simulation service platform we described in Section 2, with CityMoS [34] serving as the back-end traffic simulator. CityMoS is a high-performance, cloud-enabled, and distributed microscopic traffic simulator that is well suited for handling large-scale scenarios. The simulation follows a timestep-based approach, with each step, i.e., T_{step}, configured to 250 ms by default, as used in our experiment.

This case study comprehensively evaluates three synchronization approaches – Rigid Synchronization, the proposed buffer approach but with fixed capacity, and the proposed buffer approach with the adaptive pacing policy – in four synthetic yet generic scenarios. This section provides a detailed description of the setup and the results of the evaluation.

6.1 Experimental Design

In order to systematically explore the key characteristics of the Visualizer workload and to have a comprehensive coverage of its patterns, experimental scenarios are constructed using synthetic approaches. These scenarios are carefully designed to mimic the typical dynamics of user interactions in terms of variability and randomness.

Dynamic Visualizer Workload: This is a fundamental aspect that our synthetic scenarios must replicate, subject to the variability introduced by random user interactions. This variability is characterized by two primary dimensions: the **Computational Frequency (CF)** and the **Computational Intensity (CI)**. For example, users may alternate between a broad view, where more agents are rendered but less frequently (low CF, high CI), and a detailed view with more frequent updates but fewer vehicles (high CF, low CI). Or, users may compute different domain-specific metrics, toggling between different time windows and computational complexity.

In our experiment, we task the Visualizer with computing four specific metrics derived from a single timestep's simulation data. They include *Agent count* ($O(1)$), *Average agent speed* ($O(n)$), *Top speed per road* ($O(n \log(n)$), and *Agent distances* ($O(n^2)$). As identified in [10], these metrics are considered as reflecting the typical computational complexity in *in-situ* traffic simulation frameworks. Contrary to actual data rendering, whose computational costs can vary widely depending on rendering specifics, these metrics provide a consistent benchmark for comparison, justifying their use in our study for ease of replication. To capture the dynamics of the Visualizer's workload, we manipulate two aspects of the computation: the number of agents, which influences the CI, and the temporal frequency, which sets how often the computation occurs, i.e., influences the CF. This setup allows us to emulate the fluctuations in the Visualizer's workload similar to those in real-world scenarios, reflecting changes in CF, CI, or both.

Frequency of interactions: This is particularly crucial for our adaptive approach, which relies on sufficient time and data points to learn and adapt. While it is theoretically possible for users to make interaction requests at any time, even every simulation step, we argue that such extremely frequent interactions are rare in large-scale simulations. Meaningful analysis and knowledge extraction require a more substantial amount of time. Thus, we design the interactions generated in our synthetic scenarios to last a minimum duration of 2 minutes, i.e., 480 steps, akin to the duration of a traffic light cycle. We consider this frequency to be both representative and sufficiently challenging for our intended use case.

Simplified Simulation Workload: Typically, the workload of the simulation has less fluctuation than in the Visualizer, i.e., it remains stable for a relatively longer time. For the sake of simplicity in our experiment, we run simple random traffic simulations on a large grid road network, maintain a constant number of 60 000 simulated agents that are processed using a single thread. The number of agents can clearly scale with the number of threads, however a full scalability analysis is out of the scope of this work hence will not be further analyzed. As a result, each simulation step requires a stable amount of wall clock time to process, averaging about 75 ms.

6.2 Scenario Setup

We design four scenarios to evaluate the effectiveness of different synchronization approaches. The scenario design is to dissect and analyze the separate and combined effects of varying computational frequency (CF) and computational intensity (CI) on the Visualizer side. The Visualizer's peak workload can reach to 6 s - 8 s per step, which is about 100 times longer compare to the simulation. The computational demands of our experimental setup are comparable to those reported in previous studies [16], thus confirming the representatives of our experimental setup for similar applications in the field.

Scenario 1: Constant CF, Variable CI: The CF is fixed, i.e., querying agents every 5 s. The CI varies over five 2-minute phases. As shown in Fig. 5a, 4000 agents are queried initially, increasing to 12 000 agents in the second phase and remaining at that level through the third phase, then dropping back to 4000 for the fourth phase and continuing at this level through the fifth phase.

Scenario 2: Constant CI, Variable CF: With 25 000 computational agents fixed, the CF changes every 2 minutes in the following order: 10 s, 5 s, 2 s, and then back to 5 s and 10 s, as shown in Fig. 5b.

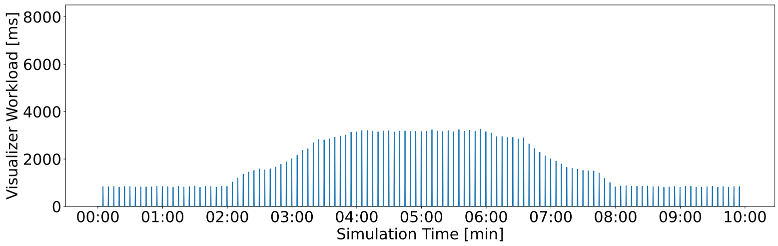
Scenario 3: Variable CF and Variable CI: Over five 2-minute phases (Fig. 5c), both CF and CI vary: Phase 1 starts with CF = 10 s and CI = 25 000 agents, Phase 2 shifts to CF = 2 s and CI = 5000 agents, Phase 3 changes to CF = 5 s and CI = 10 000 agents, then cycles back through the settings of phase 2 and 1. This scenario can effectively represent the workload fluctuations associated with multi-resolution viewing with zoom-in and zoom-out actions.

Scenario 4: Overlapped Workloads As is shown in Fig. 5d, the Visualizer processes multiple metrics simultaneously, each defined by unique CF and CI values. From 00:00 to 10:00, a load with CF = 10 s and CI = 25 000 agents is applied consistently. From 02:00 to 04:00, another load with CF = 2 s and CI = 5000 agents is added. For the rest of the time, the added load changes every 2 minutes, both in terms of CF and CI. This represents a high level of complexity that closely mirrors the diverse challenges encountered in real-world applications.

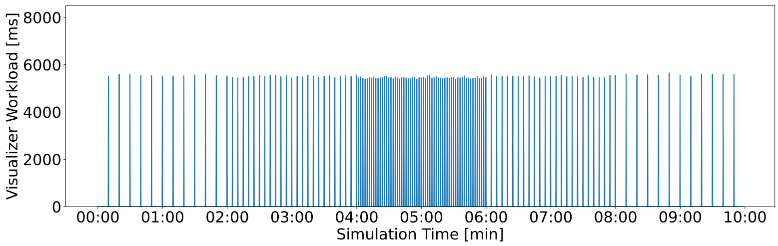
6.3 Synchronization Setup

Each scenario in our experimental framework is run with three synchronization approaches, each characterized by its own set of parameters based on the constraint of the *maximal acceptable interaction delay* ($d^{\max}$), as introduced in Section 4.

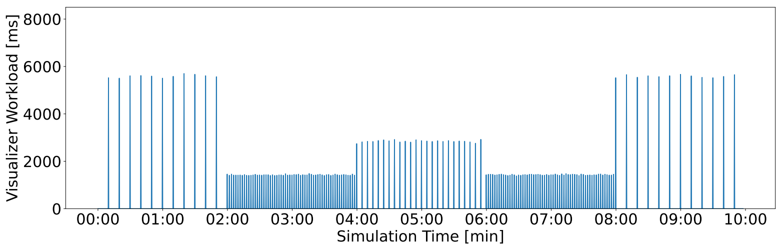
- ***Rigid***: The rigid synchronization (see Section 3.2) is used as our reference baseline. The synchronization interval T_{rigid} is determined based on $d^{\max} = 2T_{\text{rigid}} - T_{\text{step}}$, as detailed in Section 4. Notably, in alignment with backward compatibility, this approach is also implemented within our proposed architecture, setting the cycle interval $\Delta t = T_{\text{rigid}}$ and maintaining a constant single-space buffer capacity.
- ***Fixed***: The second approach in our experimental setup uses the proposed buffer-based architecture. However, in contrast to the adaptive strategy, here we adopt a simple policy: the capacity of the Bounded Buffer remains



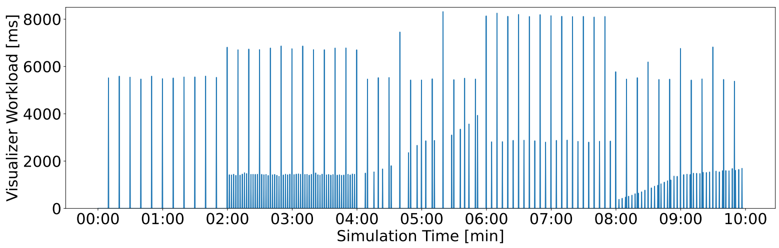
(a) Scenario 1: Constant CF, Variable CI



(b) Scenario 2: Constant CI, Variable CF



(c) Scenario 3: Variable CF, Variable CI

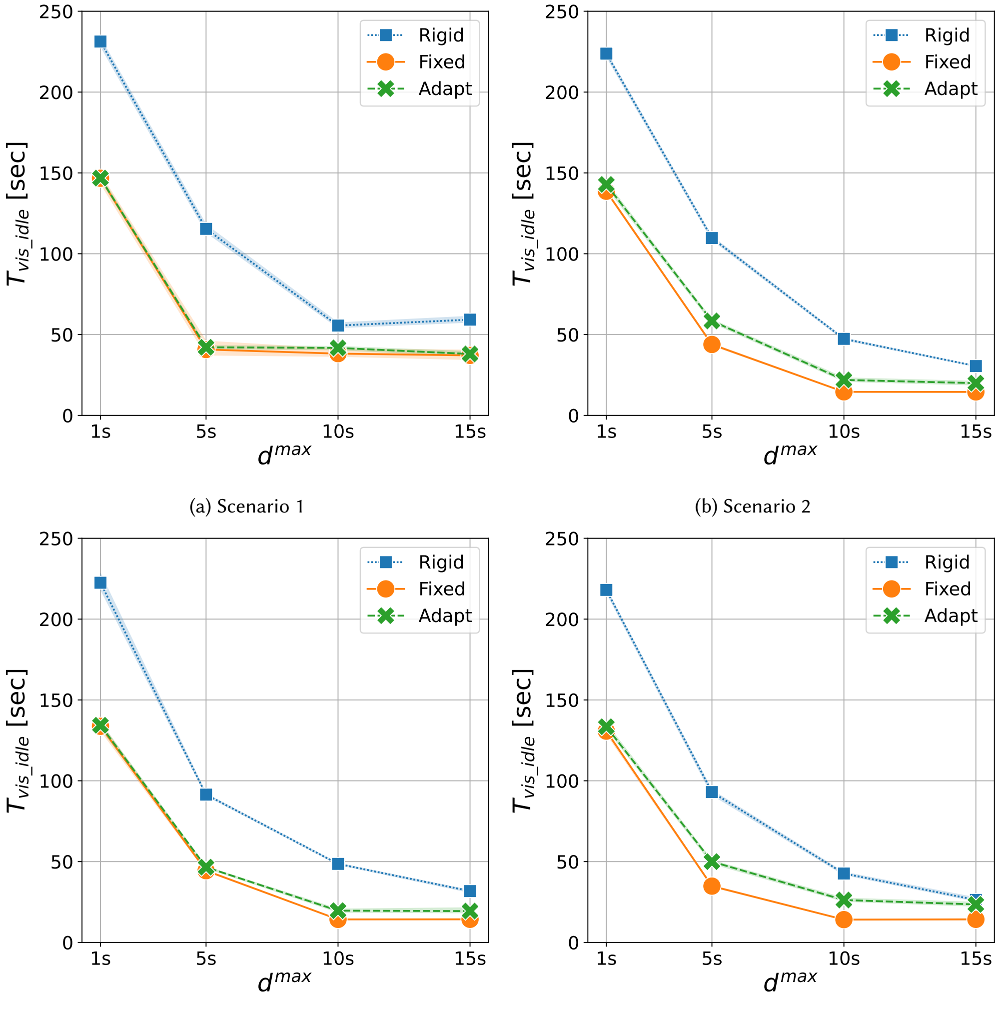


(d) Scenario 4: Overlapped Workloads

Fig. 5. Visualization workload for the tested scenarios.

unchanged for the entire duration of the run. The aim is to justify the need for adaptation and prediction. Here, the framework operates with $\Delta t = T_{\text{step}}$, and the capacity of the buffer is defined as $d^{\text{max}}/T_{\text{step}}$.

- ***Adapt***: Our comprehensive solution with the proposed adaptive pacing policy (see Section 5.4). We also set Δt to T_{step}. The maximum buffer capacity is $d^{\text{max}}/T_{\text{step}}$ and the *desired buffer lower bound*, i.e., $n_{\text{df\_low}}$ is set to 20% of it. To make predictions, we employ the *Triple Exponential Smoothing* method, which is known for its



(a) Scenario 1

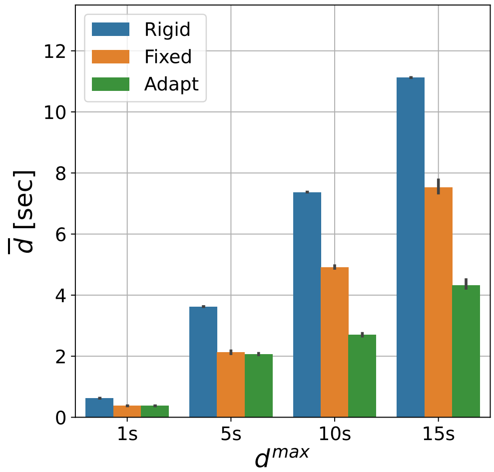
(b) Scenario 2

(c) Scenario 3

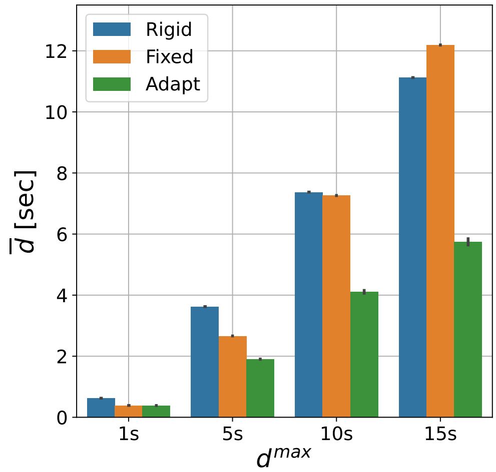
(d) Scenario 4

Fig. 6. $T_{\text{vis\_idle}}$ in various scenarios with different $d^{\max}$ for the synchronization approaches being tested.The line shadow represents the 95% confidence interval.

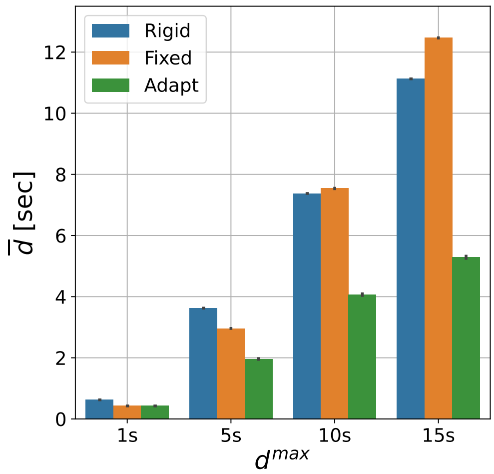
effectiveness in identifying trends and seasonality, and is also praised for its simplicity. Crucially, our prediction model relies solely on collected runtime pacing data. It does not take into account any semantic information related to the user requests. Factors such as query frequency and agent filtering are assumed to be unknown to the predictor. This assumption is intentional, as it allows us to stress test our policy to evaluate its robustness in scenarios where predictive insights are limited.



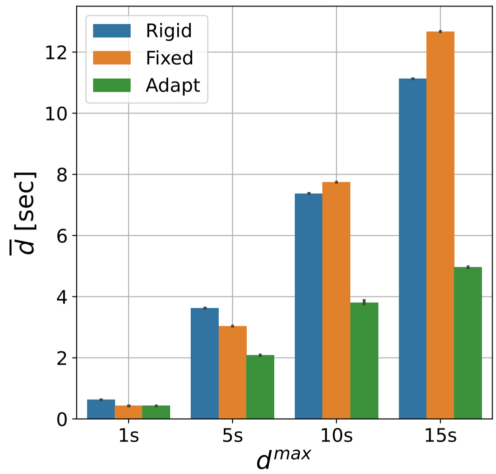
(a) Scenario 1



(b) Scenario 2



(c) Scenario 3



(d) Scenario 4

Fig. 7. $\bar{d}$ in various scenarios with different $d^{\max}$ for the synchronization approaches being tested. The error bar represents the 95% confidence interval.

6.4 Results and Analysis

The effectiveness of each synchronization method is evaluated using two metrics. The first metric (see Fig. 6) is the Visualizer Idle Time $T_{\text{vis\_idle}}$, as introduced in Section 4. Since the data processing time at the Visualizer is constant for each scenario, a lower $T_{\text{vis\_idle}}$ can also represent a shorter end-to-end time and thus a better runtime efficiency. The second metric (see Fig .7) is the average delay of interaction, i.e., $\bar{d}$. For rigid synchronization, this is calculated as $\bar{d} = \frac{3T_{\text{rigid}} - T_{\text{step}}}{2}$, as detailed in Section 4. For the buffer-based method, it is calculated by averaging the temporal gap, i.e., $n_{\text{df}} \cdot \Delta t$, of each time step between the simulation and visualization. With these two metrics, the effectiveness of each synchronization method in balancing the runtime performance and the synchronicity is evaluated, in response to the multi-objective optimization problem presented in Section 4. Each scenario is run five times to ensure the reliability of the results.

Overall, *Rigid* shows subpar performance in both $\bar{d}$ and $T_{\text{vis\_idle}}$. It records the highest value for both metrics in 13 out of 16 tests, indicating that, compared to the proposed buffer-based approach, the rigid synchronization is prone to lower interaction efficiency and weaker runtime performance. The key factor behind this performance is that the interaction delay of *Rigid* is solely dependent on the synchronization interval T_{rigid}. Unlike the buffer-based approach, where information exchange can occur at more granular intervals tailored to real-time workload conditions, *Rigid* method allows exchanges only at coarser, predetermined intervals, ignoring the actual workload dynamics. Regarding runtime performance, the rigidity and the disregard for real-time workload conditions do not ensure optimal workload balance between the Simulator and the Visualizer during each interval. This explains why an extended T_{rigid} might even lead to a decline in performance, as observed in Scenario 1 (Fig. 6a). Here, an increase of $d^{\max}$ from 10 s to 15 s (meaning a longer T_{rigid}), does not enhance runtime performance but rather worsens it, as evidenced by the increase in $T_{\text{vis\_idle}}$ from 55.56 s to 59.23 s.

Two primary conclusions can be drawn from the results of *Fixed*, i.e., using the proposed buffer-based approach with a fixed maximal capacity:

- This approach consistently outperforms others in reducing the $T_{\text{vis\_idle}}$ under the same scenarios and constraints, i.e., has the best runtime performance.
- Increasing buffer capacity within the same scenario consistently reduces $T_{\text{vis\_idle}}$ until it reaches a certain limit.

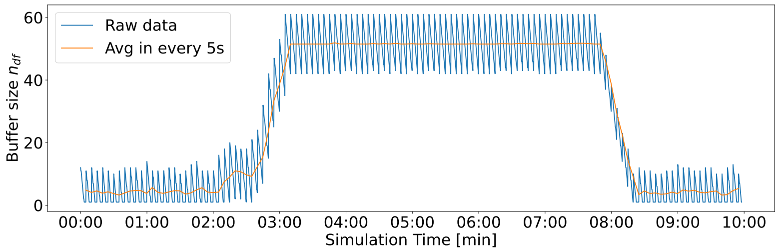
Thus, for users whose primary concern is runtime efficiency, the *Fixed* approach is a straightforward and efficient solution. However, for use cases like ours, where both runtime efficiency and synchronicity are important, this approach is not ideal. As mentioned in Section 5.2, overbuffering becomes a problem for the *Fixed* approach when the preset buffer capacity exceeds a certain limit. This results in negligible performance improvements while significantly reducing QoE. For example, in Scenario 1, raising the $d^{\max}$ from 5s to 15s, which is a threefold increase in buffer capacity, only lowers the idle time by less than 2 s (Fig. 6a), but the $\bar{d}$ increases about 3.5 times, from 2.1 s to 7.3 s (Fig. 7a). Similarly, in Scenario 2, increasing the $d^{\max}$ from 10 s to 15 s results in roughly the same $T_{\text{vis\_idle}}$ (Fig. 6b), but the average delay increases by more than 50%, i.e., from 7 s to 11.8 s (Fig. 7b). This trend is also noticeable in other scenarios.

Our proposed adaptive policy has proven effective in solving overbuffering, which has been validated in all the tested scenarios involving varying workloads for CF and CT and their combinations. Note that under the same $d^{\max}$ constraint, the maximum buffer capacity in *Adapt* is equivalent to the buffer capacity in the *Fixed* case throughout the entire run. This means that the runtime performance achieved by the *Fixed* approach serves as an upper limit that the adaptive method can achieve. However, the adaptive method distinguishes itself not only by meeting this benchmark, i.e., maintaining a close $T_{\text{vis\_idle}}$, but also by doing so with a significantly lower interaction delay.

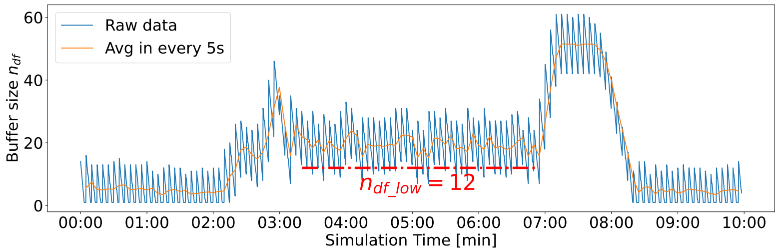
The largest difference in $T_{\text{vis\_idle}}$ between *Adapt* and *Fixed* is observed in Scenario 4 with $d^{\max}$ equals to 5 s (Fig. 6d). The adaptive approach exhibits an $T_{\text{vis\_idle}}$ of 49.99 s, which is about 15 s more than the *Fixed* approach. The main reason for this difference is the high complexity of the Visualizer workload in this case, which affects the accuracy of the predictions. On average over all cases, *Adapt* takes approximately 5 s longer than *Fixed* for the $T_{\text{vis\_idle}}$, which also reflects the difference in the overall end-to-end time. In Tab. 1 we demonstrated the speedup for all the tested scenarios with a $d^{\max}$ of 1s and 15s. As illustrated, the execution time loss of the *Adapt* case compared to *Fixed* becomes negligible when the speedup value is considered. This confirms that our suggested policy can uphold the high performance

Table 1. Speedup of the simulation of examined scenarios. The speedup is determined by the proportion of the simulation's wall clock duration to the physical time (i.e., 10 min).

$d^{\max}$	Approach	Scenario 1	Scenario 2	Scenario 3	Scenario 4
1 s	Rigid	1.40	0.66	1.06	0.80
	Fixed	1.73	0.73	1.25	0.91
	Adapt	1.73	0.72	1.25	0.90
15 s	Rigid	2.29	0.82	1.58	1.07
	Fixed	2.43	0.84	1.61	1.08
	Adapt	2.43	0.83	1.59	1.07



(a) Fixed



(b) Adapt

Fig. 8. The fluctuations in n_{df} during the entire simulation in scenario 1, with the $d^{\max}$ set to 15 seconds. The interaction delay at any time equals $n_{\text{df}} \times 250$ ms.

capability of the buffer-based approach compared to the *Rigid* approach. In terms of the improved synchronicity, fig. 7 shows that *Adapt* consistently has the lowest $\bar{d}$ compared to other approaches. For example, with a $d^{\max}$ of 15 s, the $\bar{d}$ using the adaptive method ranges from 39% to 57% of that using the fixed capacity approach. However, it is also observed that with the increase of $d^{\max}$ from 10 s to 15 s in *Adapt*, there is minimal improvement in the $T_{\text{vis\_idle}}$ but a more pronounced increase in the $\bar{d}$, similar to what is seen with the *Fixed* approach. This phenomenon is largely attributed to the increase in the *desired buffer lower bound* ($n_{\text{df\_low}}$) as part of our setup. When compared to *Fixed*, the increase in $\bar{d}$ within the adaptive method is notably less substantial.

A detailed comparative analysis between *Fixed* and *Adapt* is facilitated by Fig. 8, which illustrates the variation of the number of data frames (n_{df}) within the buffer throughout the entire simulation in Scenario 1, with the $d^{\max}$ set to 5 s. It can be observed that during the first phase, from 00:00 to 02:00, the buffer in the *Fixed* approach (Fig. 8a) is consistently emptied before starting to refill. This is due to the relatively low CI of the visualization during this phase (Fig. 5a), which allows the Visualizer to consume data faster than the Simulator produces at each 5-second CF. To ensure a minimal $T_{\text{vis\_idle}}$, it is crucial for the simulation to operate at its full speed. The *Fixed* approach achieves this naturally, as the buffer never reaches its full capacity, thus facilitating the continuous activation of the simulation. Notably, *Adapt* (Fig. 8b) also meets this objective, as it keeps the n_{df} at the same level as the fixed approach, indicating the success of the proposed adaptive policy in making the right decision.

As the visualization workload increases, it cannot fully process the incoming simulation data in each computation interval. In *Fixed*, this leads to an incremental accumulation of unprocessed df(s) in the buffer (from 03:00 to 03:30), causing the buffer to reach its capacity limit and eventually causing the overbuffering problem. From 03:30 to 07:00, the buffer is consistently full, which in turn leads to significant temporal gaps between the simulation and visualization. In contrast, the adaptive approach successfully maintains the buffer size at the minimum necessary level, dictated by the policy's $n_{\text{df\_low}}$, which is set at 12, the equivalent of a 3 s time difference. During the period from 03:30 to 07:00, the buffer size (n_{df}), as shown by the orange line in Fig. 8b, remains relatively stable. This stability suggests that the simulation is adeptly synchronized to maintain a steady temporal gap with the visualization and matches its pace. Therefore, the adaptive policy not only guarantees an uninterrupted flow of data to the Visualizer – preventing any idle time by ensuring that the buffer is never empty – but also holds significantly less data in the buffer compared to the fixed approach, resulting in superior QoE.

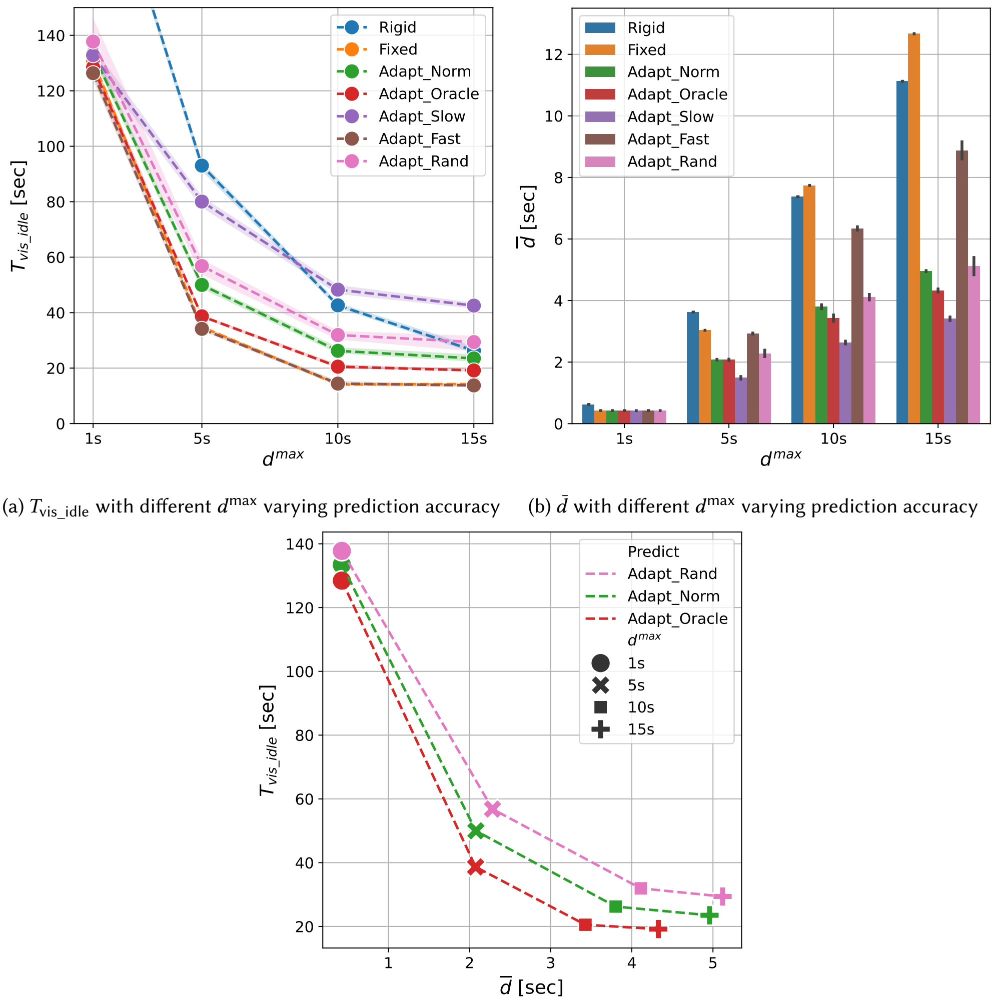
Between 07:00 and 08:00, as the Visualizer's processing speed accelerates and exceeds that of the simulation, all of the data stored in the buffer is consumed under the *Fixed* approach. In the *Adapt* case, the Pacing Controller accurately anticipates a potential data shortage. To counteract this, it proactively increases the buffer capacity to the upper limit and allows the simulation to run at full speed in advance. This forward-thinking strategy leads to a temporary increase in buffer storage around 07:30 (Fig. 8b), thus ensuring that the best possible runtime performance is maintained, as with the *Fixed* approach.

In summary, the results of this detailed analysis of buffer status clearly validate the adaptive approach's ability to dynamically adjust the buffer capacity and pacing of the simulation in response to workload dynamics. Compared to the fixed buffer capacity approach, this adaptability plays a crucial role in preventing overbuffering. And it still ensures optimal runtime efficiency by proactively increasing the simulation speed in anticipation of potential simulation slowdowns. These results are consistent with our initial expectations for the designed policy, and demonstrate its effectiveness in striking a delicate balance between runtime efficiency and synchronicity in an interactive simulation system.

6.5 Discussion on Prediction Accuracy

In this section, we investigate the impact of prediction accuracy on our synchronization method, focusing on how inaccurate predictions can affect overall system performance. We have specifically selected Scenario 4 (refer to Section 6.3) for our experiments due to its complex and dynamic Visualizer workload characteristics. As the simulator itself consistently performs stably over time, making it easier to be predicted compared to the Visualizer, our analysis here concentrates solely on the prediction accuracy of the Visualizer.

To examine the effects of prediction errors, synthetic noise is added to the workload predictions of the Visualizer, and we analyzed how these errors affected the decision-making process, ultimately influencing both system runtime efficiency (Fig. 9a) and user QoE (Fig. 9b). For a baseline comparison, we referred to outcomes from the prior section using *Triple Exponential Smoothing* predictions as ***Adapt_Norm***, depicted in Fig. 6d and Fig. 7d. The first test case, ***Adapt_Oracle***, actually involves no real prediction. Instead, we use recorded historical ground-truth data directly for the policy's decision making. Hence, *Adapt_Oracle* should represent the highest possible accuracy. Following this, we



(a) $T_{\text{vis\_idle}}$ with different $d^{\max}$ varying prediction accuracy

(b) $\bar{d}$ with different $d^{\max}$ varying prediction accuracy

(c) Pareto front minimizing both $T_{\text{vis\_idle}}$ and $\bar{d}$ across varying prediction accuracy

Fig. 9. Experiment outcomes on the effects of prediction accuracy

introduce three different cases with varying kinds of prediction noise. They modified the original prediction outcomes p, i.e., Visualizer execution time for a step, to p' using different functions:

- ***Adapt_Slow:*** In this setup, each predicted Visualizer's execution time is intentionally increased by a random percentage, i.e., $p' = p * (1 + U(0, 1))$, where U is a continuous uniform distribution. This will effectively bias the prediction data, simulating a slower-than-actual performance of the Visualizer.
- ***Adapt_Fast***: Conversely, this configuration reduces the predicted time cost of the Visualizer by a random percentage, i.e., $p' = p * (1 + U(-1, 0))$, thereby tending to forecast a faster Visualizer than it actually is.

- ***Adapt_Rand***: In this case, the predicted times are modified randomly i.e., $p' = p * (1 + U(-1, 1))$. This adds noise to the prediction without creating a bias to the mean of the original predicted dataset.

Initially, with a low constraint (i.e., $d^{\max} = 1$ s), the prediction inaccuracies do not significantly alter the policy's performance concerning the two metrics, i.e., $T_{\text{vis\_idle}}$ (see Fig. 9a) and $\bar{d}$ (see Fig. 9b). This is because the buffer's maximum capacity is small (i.e., 4), limiting the policy's ability to demonstrate its effectiveness. However, as the $d^{\max}$ increases beyond 5 s (buffer with a maximum capacity exceeding 20), the impact of prediction errors becomes more evident.

As anticipated, we observe the following from *Adapt_Slow* and *Adapt_Fast*: *Adapt_Slow* tends to underestimate the execution performance of the Visualizer, predicting a slower speed than the actual. Consequently, the simulation is paced more slowly to align with the Visualizer. This resulted in the longest Visualizer Idle Time, i.e., the longest end-to-end time compared to other prediction cases. As shown in Fig. 9a, represented by the purple dashed line, when the $d^{\max}$ exceeds 10 s, the $T_{\text{vis\_idle}}$ of *Adapt_Slow* exceeds even that of *Rigid*. However, it has the advantage of the shortest interaction delays (refer to the purple bar in Fig. 9b), signifying the best QoE. Conversely, *Adapt_Fast* tends to overestimate the Visualizer's capabilities, prompting simulations to run too quickly in an attempt to catch up with the anticipated performance of the Visualizer. This results in the shortest $T_{\text{vis\_idle}}$ among all adaptive scenarios, nearly matching the upper limit of the *Fixed* case (see the overlapping brown and yellow dashed lines in Fig. 9a). However, not surprisingly, this approach incurs the highest interaction delay among all adaptive scenarios (brown bar in Fig. 9b), though it remains lower than those observed in *Fixed* and *Rigid*.

Comparing *Adapt_Oracle*, *Adapt_Norm*, and *Adapt_Rand*, it is clear that the closeness of predictions to the ground truth—ranging from most accurate to least—is directly influences the quality of outcomes. Fig. 9c shows that more accurate predictions result in improved results, as evidenced by a lower Pareto front. This suggests that under the identical constraints, more precise predictions is able to effectively minimize both metrics, $T_{\text{vis\_idle}}$ and $\bar{d}$, at the same time.

To sum up, this study underscores the consequences of slower or faster predictions and the influence of random noise on our policy's decision-making processes. The findings validate our design principles, demonstrating that the framework operates as intended and suggests that improved prediction accuracy could boost the performance of our proposed policy.

7 CONCLUSION AND FUTURE WORK

In this paper, we contribute to the advancement of Visual Interactive Simulation (VIS) synchronization in three respects.

First, we define the enhancement of simulation runtime efficiency and synchronicity as a multi-objective optimization challenge (see Section 4). This perspective provides a structured approach to addressing the complexities inherent in VIS synchronization. Second, we present a novel framework that utilizes a *Controller* as a drop-in replacement for a straightforward Visualizer-Simulator connection, making it applicable to a wide range of applications. Third, we introduce a heuristic algorithm that utilizes predictive workload analysis to dynamically control the pace of the simulation to match the real-time workload of the visualization. Our experimental evaluation demonstrates that this method substantially enhances runtime efficiency and Quality of Experience (QoE) compared to traditional fixed-interval synchronization.

As future research, we consider to integrate our synchronization strategy with resource allocation methods. By dynamically allocating computational resources to simulation and visualization tasks to regulate their processing speed, we can potentially improve performance beyond what our current buffer-based policy allows. This integration could provide a comprehensive solution for balancing the runtime efficiency and synchronicity in VIS systems.

REFERENCES

[1] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. 2015. ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In *1st Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM, 25–29. https://doi.org/10.1145/2828612.2828624

[2] Andrew C Bauer, Hasan Abbasi, James Ahrens, Hank Childs, Berk Geveci, Scott Klasky, Kenneth Moreland, Patrick O'Leary, Venkatram Vishwanath, Brad Whitlock, et al. 2016. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Computer Graphics Forum* 35, 3 (June 2016), 577–597. https://doi.org/10.1111/cgf.12930

[3] John Biddiscombe, Jerome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. 2011. Parallel Computational Steering and Analysis for HPC Applications using a ParaView Interface and the HDF5 DSM Virtual File Driver. In *Eurographics Symposium on Parallel Graphics and Visualization*. Eurographics Association, 91–100. https://doi.org/10.2312/EGPGV/EGPGV11/091-100

[4] Aradhya Biswas and Richard Fujimoto. 2018. Zero Energy Synchronization of Distributed Simulations. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 85–96. https://doi.org/10.1145/3200921.3200938

[5] Stefan Boschert and Roland Rosen. 2016. Digital Twin—The Simulation Aspect. *Mechatronic futures: Challenges and solutions for mechatronic systems and their designers* (2016), 59–74. https://doi.org/10.1007/978-3-319-32156-1_5

[6] Marc Buffat, Anne Cadiou, Lionel Le Penven, and Christophe Pera. 2016. In situ analysis and visualization of massively parallel computations. *The International Journal of High Performance Computing Applications* 31, 1 (July 2016), 83–90. https://doi.org/10.1177/1094342015597081

[7] Gerasimos Chourdakis, Kyle Davis, Benjamin Rodenberg, Miriam Schulte, Frédéric Simonis, Benjamin Uekermann, Georg Abrams, Hans-Joachim Bungartz, Lucia Cheung Yau, Ishaan Desai, Konrad Eder, Richard Hertrich, Florian Lindner, Alexander Rusch, Dmytro Sashko, David Schneider, Amin Totounferoush, Dominik Volland, Peter Vollmer, and Oguz Ziya Koseomur. 2022. preCICE v2: A sustainable and user-friendly coupling library. *Open Research Europe* 2 (Sept. 2022). https://doi.org/10.12688/openreseurope.14445.2

[8] Frederica Darema. 2011. DDDAS Computational Model and Environments. *Journal of Algorithms & Computational Technology* 5, 4 (Dec. 2011), 545–560. https://doi.org/10.1260/1748-3018.5.4.545

[9] Matthieu Dorier, Robert Sisneros, Leonardo Bautista Gomez, Tom Peterka, Leigh Orf, Lokman Rahmani, Gabriel Antoniu, and Luc Bougé. 2016. Adaptive Performance-Constrained In Situ Visualization of Atmospheric Simulations. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 269–278. https://doi.org/10.1109/CLUSTER.2016.25

[10] Xiaorui Du, Zhuoxiao Meng, Anibal Siguenza-Torres, Alois Knoll, Adriano Pimpini, Andrea Piccione, Stefano Bortoli, and Alessandro Pellegrini. 2023. Autonomic Orchestration of in-Situ and in-Transit Data Analytics For Simulation Studies. In *2023 Winter Simulation Conference (WSC)*. IEEE, 781–792. https://doi.org/10.1109/WSC60868.2023.10408191

[11] Richard Fujimoto, Joseph Barjis, Erik Blasch, Wentong Cai, Dong Jin, Seunghan Lee, and Young-Jun Son. 2018. Dynamic data driven application systems: research challenges and opportunities. In *2018 Winter Simulation Conference (WSC)*. IEEE, 664–678. https://doi.org/10.1109/WSC.2018.8632379

[12] Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. Co-Simulation: A Survey. *ACM Computing Surveys (CSUR)* 51, 3 (May 2018), 1–33. https://doi.org/10.1145/3179993

[13] Maria Hybinette and Richard Fujimoto. 1997. Cloning: a novel method for interactive parallel simulation. In *29th Conference on Winter Simulation (WSC)* (Atlanta, Georgia, USA). ACM, 444–451. https://doi.org/10.1145/268437.268523

[14] Christopher Johnson, Steven G Parker, Charles Hansen, Gordon L Kindlmann, and Yarden Livnat. 1999. Interactive simulation and visualization. *Computer* 32, 12 (1999), 59–65. https://doi.org/10.1109/2.809252

[15] Yi Ju, Adalberto Perez, Stefano Markidis, Philipp Schlatter, and Erwin Laure. 2022. Understanding the Impact of Synchronous, Asynchronous, and Hybrid In-Situ Techniques in Computational Fluid Dynamics Applications. In *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 295–305. https://doi.org/10.1109/eScience55777.2022.00043

[16] Takuma Kawamura, Yuta Hasegawa, and Yasuhiro Idomura. 2023. Interactive steering on in situ particle-based volume rendering framework. *Journal of Visualization* 27, 1 (Sept. 2023), 89–107. https://doi.org/10.1007/s12650-023-00945-z

[17] James Kress, Scott Klasky, Norbert Podhorszki, Jong Choi, Hank Childs, and David Pugmire. 2015. Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay. In *1st Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM. https://doi.org/10.1145/2828612.2828623

[18] Marcel Krüger, Simon Oehrl, Ali C. Demiralp, Sebastian Spreizer, Jens Bruchertseifer, Torsten W. Kuhlen, Tim Gerrits, and Benjamin Weyers. 2022. Insite: A Pipeline Enabling In-Transit Visualization and Analysis for Neuronal Network Simulations. In *ISC High Performance 2022*. Springer, 295–305. https://doi.org/10.1007/978-3-031-23220-6_20

[19] Henry Lehmann and Bernhard Jung. 2014. In-situ multi-resolution and temporal data compression for visual exploration of large-scale scientific simulations. In *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE, 51–58. https://doi.org/10.1109/LDAV.2014.7013204

[20] Feng Li and Fengguang Song. 2023. INSTANT: A Runtime Framework to Orchestrate In-Situ Workflows. In *European Conference on Parallel Processing*. Springer, 199–213. https://doi.org/10.1007/978-3-031-39698-4_14

[21] Jianping Kelvin Li, Misbah Mubarak, Robert B. Ross, Christopher D. Carothers, and Kwan-Liu Ma. 2017. Visual Analytics Techniques for Exploring the Design Space of Large-Scale High-Radix Networks. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 193–203. https://doi.org/10.1109/CLUSTER.2017.26

[22] Bryan Lim and Stefan Zohren. 2021. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A* 379, 2194 (Feb. 2021), 20200209. https://doi.org/10.1098/rsta.2020.0209

[23] Zhuoxiao Meng, Anibal Siguenza-Torres, Mingyue Gao, Margherita Grossi, Alexander Wieder, Xiaorui Du, Stefano Bortoli, Christoph Sommer, and Alois Knoll. 2023. Towards Discrete-Event, Aggregating, and Relational Control Interfaces for Traffic Simulation. In *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '23)*. ACM, 12–22. https://doi.org/10.1145/3573900.3591116

[24] Anirudh Modi, Lyle N. Long, and Paul E. Plassmann. 2002. Real-Time Visualization of Wake-Vortex Simulations Using Computational Steering and Beowulf Clusters. In *5th International Conference on High Performance Computing for Computational Science (VECPAR 2002)* (Porto, Portugal). Springer, 464–478. https://doi.org/10.1007/3-540-36569-9_31

[25] S. Narayanan and Phani Kidambi. 2011. *Interactive Simulations: History, Features, and Trends.* Springer London, London. 1–13 pages. https://doi.org/10.1007/978-0-85729-883-6_1

[26] Robert M. O'Keefe. 1987. What is visual interactive simulation? (and is there a methodology for doing it right?). In *19th conference on Winter simulation (WSC '87)* (Atlanta, Georgia, USA). ACM, 461–464. https://doi.org/10.1145/318371.318635

[27] Steven G. Parker and Christopher R. Johnson. 1995. SCIRun: a scientific programming environment for computational steering. In *1995 ACM/IEEE Conference on Supercomputing* (San Diego, California, USA). ACM. https://doi.org/10.1145/224170.224354

[28] Kalyan Perumalla, Peter Barnes, Maximilian Bremer, Kevin Brown, Cy Chan, Stephan Eidenbenz, K. Scott Hemmert, Adolfy Hoisie, Benjamin Newton, James Nutaro, Tomas Oppelstrup, Robert Ross, Markus Schordan, and Nathan Urban. 2022. *Computer Science Research Needs for Parallel Discrete Event Simulation (PDES).* Technical Report. Office of Scientific and Technical Information (OSTI). https://doi.org/10.2172/1855247

[29] Zhe Wang, Matthieu Dorier, Pradeep Subedi, Philip E. Davis, and Manish Parashar. 2023. Adaptive elasticity policies for staging-based in situ visualization. *Future Generation Computer Systems* 142 (May 2023), 75–89. https://doi.org/10.1016/j.future.2022.12.010

[30] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. 2011. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Eurographics Symposium on Parallel Graphics and Visualization*, Torsten Kuhlen, Renato Pajarola, and Kun Zhou (Eds.). The Eurographics Association. https://doi.org/10.2312/EGPGV/EGPGV11/101-109

[31] Haowen Xu, Chieh (Ross) Wang, Anne Berres, Tim LaClair, and Jibonananda Sanyal. 2021. Interactive Web Application for Traffic Simulation Data Management and Visualization. *Transportation Research Record: Journal of the Transportation Research Board* 2676, 1 (Aug. 2021), 274–292. https://doi.org/10.1177/03611981211035760

[32] Hongfeng Yu, Tiankai Tu, Jacobo Bielak, Omar Ghattas, Julio López, Kwan-Liu Ma, David R. O'Hallaron, Leonardo Ramirez-Guzman, Nathan Stone, Ricardo Taborda-Rios, and John Urbanic. 2006. Remote Runtime Steering of Integrated Terascale Simulation and Visualization. (2006). https://doi.org/10.1184/R1/6608987.v1

[33] Daniel Zehe, Alois Knoll, Wentong Cai, and Heiko Aydt. 2015. SEMSim Cloud Service: Large-scale urban systems simulation in the cloud. *Simulation Modelling Practice and Theory* 58 (Nov. 2015), 157–171. https://doi.org/10.1016/j.simpat.2015.05.005

[34] Daniel Zehe, Suraj Nair, Alois Knoll, and David Eckhoff. 2017. Towards CityMoS: A Coupled City-Scale Mobility Simulation Framework. (April 2017).

[35] Daniel Zehe, Vaisagh Viswanathan, Wentong Cai, and Alois Knoll. 2016. Online Data Extraction for Large-Scale Agent-Based Simulations. In *2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '16)*. ACM, 69–78. https://doi.org/10.1145/2901378.2901384

[36] Hongbo Zou, Fang Zheng, Matthew Wolf, Greg Eisenhauer, Karsten Schwan, Hasan Abbasi, Qing Liu, Norbert Podhorszki, and Scott Klasky. 2012. Quality-Aware Data Management for Large Scale Scientific Applications. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 816–820. https://doi.org/10.1109/SC.Companion.2012.114