Holes and Clashes in Simulated Soccer: When a Real-Time Simulation System Meets Compute-Hungry Agents

Thomas Gabel and Eicke Godehardt

Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences
60318 Frankfurt am Main, Germany
{tgabel|godehardt}@fb2.fra-uas.de

Abstract. The paper addresses the challenges and implications of (lacking) synchronization between agents in real-time multi-agent simulation systems. Based on two specific manifestations of mis-synchronization in 2D Soccer Simulation, termed holes and clashes, the paper makes two contributions: First, it provides a thorough historical review of the problem's existence and urgency of occurrence, covering the time from the very early beginnings of the RoboCup initiative in the late 1990s till today. Second, it presents the results of an exhaustive empirical study that reveals quantitatively what impact synchronization problems may have on the playing performance of a simulated soccer team.

1 Introduction

Our story begins with a match of our soccer simulation team FRA-UNIted against the team Tehran during RoboCup 2022. Using flip book-style, Fig. 1 visualizes one peculiar scene in which our team's agent #2 (blue) performed a failing tackle at $t = 4792$ in a situation where the tackle success probability was zero, whereas it would have been quite promising to perform that tackle a time step earlier, i.e. at $t = 4791$ where its chance of success would have been 84.6%.

Knowing that our team's calculation for the tackle decision is efficient and fail-safe, the observed behavior seemed miraculous. Inspecting the game's logfile revealed that #2 intended to perform its tackling at $t = 4791$, but that the corresponding command *arrived late*, i.e. not until the succeeding time step, followed by a turn command which was intended for $t = 4792$. That situation is called a *hole* (no command at $t = 4791$) with a subsequent *clash* of two commands, where the soccer simulator executes only the first of the commands arrived and disregards the latter.

Given that the situation took place near our team's penalty area and having agent #2 forced to ten cycles of inactivity after its failed tackle, the opponent team scored a goal. Challenged by this instance, we decided to research on the topic of holes and clashes in soccer simulation 2D in more depth. This paper presents our findings: In Section 2, we start by introducing some necessary notation and discuss related work. Section 3 provides a historical review on the

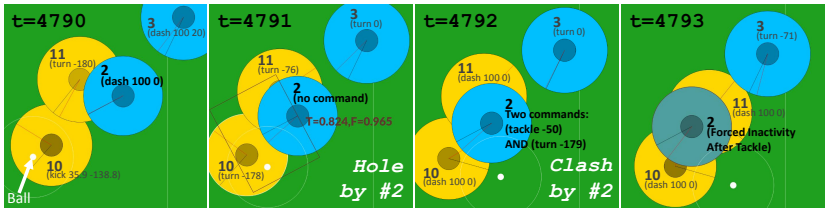

Fig. 1. Tracking the Sequence of Actions of Player #2: After a dash with 100% power into direction 0°, the rectangle inside player #2 at $t = 4791$ indicates a tackle success probability of $T = 0.824$. Instead of doing the tackle, however, #2 issues no command at all – a hole arises. In the subsequent cycle $t = 4792$, two commands from #2 arrive at the server – a clash. While the latter command is ignored by the server, the former one, a tackle into direction -50°, is executed. Though this would have been a good action at $t = 4791$, it is doomed to fail at $t = 4792$ since now the ball is already so distant that the tackle success probability is zero. After the tackle has failed, #2 is, according to the rules of the simulation, forced to 10 steps of inactivity starting at $t = 4793$ which allow the opponent players #10 and #11 to dribble ahead.

occurrence of holes and clashes covering the entire history of the RoboCup initiative. Finally, in Section 4 we analyze and show empirically how significant the impact of clashes is on a team's soccer-playing performance.

2 Foundations

In RoboCup's 2D Simulation League, two teams of simulated soccer-playing agents compete against one another using the Soccer Server [12] as real-time soccer simulation system which facilitates playing soccer in a client/server-based style. It simulates the playing field, communication, the environment and its dynamics, while the player clients connect to the server and send their intended actions once per simulation cycle to the server. The server takes all agents' actions into account, computes the subsequent world state and provides all agents with information about their environment via appropriate messages over UDP.

Decision making must be performed in real-time or, more precisely, in discrete time steps: Every 100ms the agents can execute a basic action and the world-state will change based on the individual actions of all players. The set of available actions include dash, turn, kick, tackle, and some other types of action each of which can be parameterized by some action or direction argument. Most importantly, only one action can be executed per time step which is why these basic actions must be combined cleverly in consecutive time steps in order to create "higher-level actions" like intercepting balls or doing dribblings, and that missing one of them in a sequence may hamper an agent's overall intentions.

2.1 Notation

In the early days of RoboCup's soccer simulation it was a challenge to assure synchronization between clients and server. On the one hand, the UDP protocol

used for exchanging messages did not guarantee the arrival of some packet at the target implying that the agents' states were eventually not synchronized with the server cycle. On the other hand, games are run under real-time constraints and, depending on the particular hardware employed at a RoboCup tournament's venue, the agents' computational thirst at times exceeded the available resources often resulting in messages to arrive too late at the server's side.

Definition of Holes and Clashes
In its nowadays used setup, each agent is allowed to send a single body-related command to the Soccer Server in each time step. We define a *Hole* as a situation where no such body-related command has arrived at the Soccer Server from an agent. Thus, there can be up to eleven holes per team per step. And given that the default duration of a match is ten minutes (i.e. 6000 simulation cycles), there might be up to 66.000 holes per team per match.

Generally, the occurrence of a hole *may* indicate a potential problem, but it does not necessarily have to. While a hole certainly represents an agent's missed opportunity to act, there might also be times where it intentionally sends no command, if, for example, it sees no need to act.

A *Clash* denotes the situation where the Soccer Server receives more than only one body-related command within one simulation cycle from one specific agent. Variations in the speed with which a network packet containing a command is transmitted used to be an erstwhile cause for clashes. Today's main cause for clashes are compute-hungry agents or, stated differently, underpowered simulation machines. If an agent's calculations for determining its next action takes longer than 100ms and, thus, then the command cannot arrive at the Soccer Server within the current simulation cycle, but will so when the next one has already started. If, moreover, the agent's subsequent action arrives in time, two commands will have arrived within the time step's 100ms time window in which case the Soccer Server executes the former and discards the latter.

While a hole *may* indicate some problem, a *clash* certainly does so as one of the commands issued by the agent will be dropped. An overview of a selection of situations in which holes and clashes occur, is provide in Fig. 2. The flipbook example from RoboCup 2022 visualized in Fig. 1 represents a classic clash.

2.2 Related Work

Time synchronization in distributed systems is a prominent research topic, notably in the areas of (wireless) sensor networks, internet of things and distributed computing [10], but of course also in the domain of robotics and robotic soccer [13, 8]. In Soccer Simulation 2D the task of synchronizing the soccer-playing clients with the server was a major challenge in its first decade and was discussed frequently in the literature. In [7], a comparative study is presented on different synchronization mechanisms between players and the environment, differences among which are attributed to the occurrence of holes and clashes which are detected in a match's log file in a similar manner as we do. Different authors

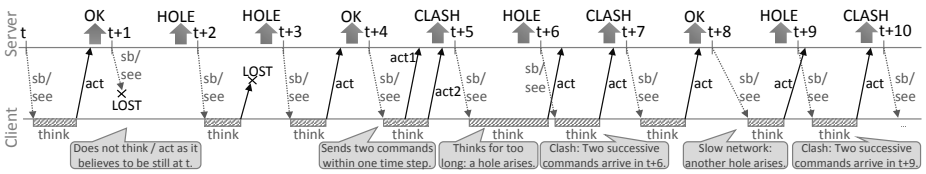

Fig. 2. The Soccer Server maintains the beat by sending "sense body" (sb) and "see" messages to clients at the beginning of a cycle. These messages are received sooner or later by an agent, depending on the network, or, in extreme cases, may get lost $(t+1)$. When received, the agent takes this as the start of the next time step, incorporates the transmitted data into its world model, starts calculating its next action and, finally, sends it to the server. The required calculation (think) should be finished fast (i.e. in less than 100ms) to ensure that the act message arrives at the server within the current cycle. Holes occur when messages got lost $(t+2)$ or when an agent's command arrives too late at the server, either because of a slow network $(t+8)$ or due to too demanding computations by the agents, finished too late for the message to arrive at the server within the current cycle $(t+5)$. Clashes may occur if it (erroneously) sends multiple commands within a single cycle $(t+4$, seldomly observed in practice). If, however, a clash occurs immediately after a hole (sometimes also called a "black hole"), then most certainly something "is wrong" with the network, the timing, the computational demands, or generally with the synchronization of server and clients $(t+6$ and $t+9)$.

attempted to find counter strategies to get along with synchronization offsets [11] or took a formal approach to overcome the linear stochastic asynchronous control problem in simulated soccer [14].

In later years, the focus shifted away from holes and clashes mainly because synchronization became less of an issue due to matured network technology and faster machines. Besides, version 12 of the Soccer Server (2008) introduced a so-called synchronized viewing mode which, effectively, abolished the concept of asynchronously sending visual information and sense-body perception [7] since all teams switched to the new, simpler mode. A brief, yet general review of the development of the league and its changing foci can be found in [15].

Many discussions in the simulation community's veteran `robocup-sim` mailing list referred to holes and clashes, as well. Yet, the second-to-last thread in that list with reference to the topic at hand originates from 2007: ATHumboldt's former team leader R. Berger reports on an "abnormal number of holes and clashes in [...] qualification games", "found a correlation between the size of the team logfile that is recorded by the [league manager] script and the mentioned synchronization problems", and linked "these effects [to] logging to a non-local NFS-imported directory". Ever since, extensive logging by teams has been forbidden in general during tournaments. The last thread on that topic in the mentioned mailing list is from 2023 by the first author of this paper [9] about a noticeable reoccurrence of holes and clashes in 2022's RoboCup tournament which we summarize below. Finally, in preparation of the RoboCup 2023 tournament, a thorough analysis of the situation during recent tournaments has been conducted by O. Amini (Cyrus) and A. Rad and published online [3].

3 Historical Review

In the first years after the start of the RoboCup initiative in 1996, soccer simulation teams primarily fought with unreliable network connections between server and host machines and with delayed or dropped packets that resulted in both, holes and clashes. As network technology became more reliable, around 2001/02 the focus started to shift towards computational issues and the task of guaranteeing to cease an agent's inferencing and calculations within 100ms of time.

Methodology We performed an extensive techno-archaeological study to investigate the question how many holes and clashes occurred during historic RoboCup events. For this, we retrieved all historical RoboCup world championships log files which are available online on `chaosscripting.net` (covering the very beginnings in 1996 and ceasing in 2017) as well as on `robocup.info` (covering competitions from 2004 till the very recent tournament in 2023) [1]. In total, we focused on the almost 3400 official world championships matches that took place on-site in the previous 28 years. We had to skip a few of the very early years where the log file format was subject to heavy changes and was preserved only in binary format where it would have meant enormous effort to analyze and interpret the artifacts of that time with today's tools.

In the first years our study covers, holes were prevalent (e.g. the 1996 tournament saw more than 1.6 million holes in total), but their frequency of occurrence diminished over the years (e.g. from 6.600 holes per team per match on average during RoboCup 2001 in Seattle to 32 during the RoboCup 2023 in Bordeaux, cf. Fig. 3, top). Yet, these numbers are a volatile basis for further analyses, since holes are counted, too, if some player erroneously disconnects or gets trapped in an endless loop due to a programming bug and, henceforth, sends no more commands at all. Therefore, all our further analyses focus exclusively on clashes occurring immediately after holes (aka black holes) as this combination hints most likely to problems in timing and synchronization of agents and the server.

Even when focusing on clashes solely, it is hard to draw general conclusions because team-internal problems and bugs might distort any statistic on clashes. We therefore conducted an analysis to identify outliers, i.e. teams whose binaries performed extraordinarily conspicuously in terms of producing holes and clashes, using the local outlier factor (LOF) algorithm [6]. Although it is not guaranteed that all teams with severe internal problems are identified this way, we empirically found that several "beknown" outliers were successfully filtered out using this approach, including Damavand'23 (heavy problems on the last tournament day), FRA-UNIted'18 (mistakenly left logging enabled in first rounds), or problems of Java-based teams like KickOffTUG'09 to name just a few.

Discussion Essentially, when observing multiple teams with a moderate number of clashes within one season's tournament, this is a strong indication that there is or, better, was a general issue with the competition set-up. Fig. 3 (bottom) visualizes our findings: We report both, the number of clashes per team per match averaged over all teams (with and without outliers) as well as this value

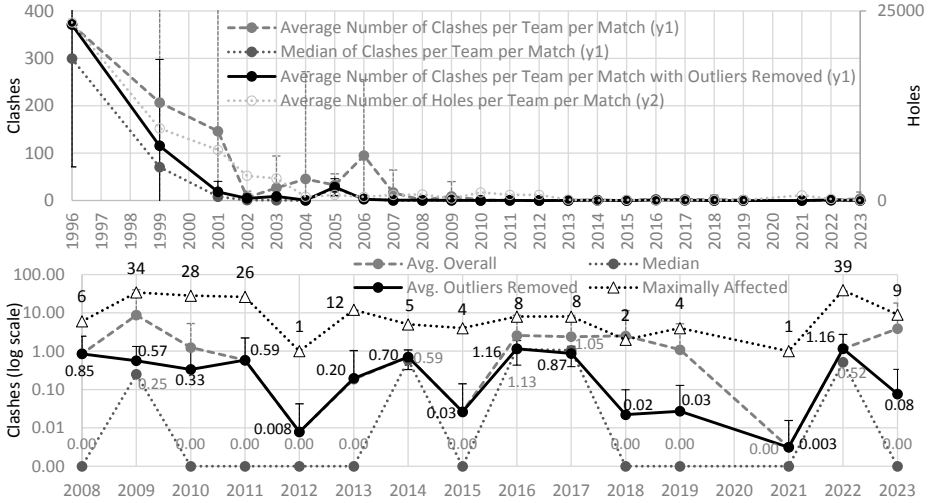

Fig. 3. Historical Review on the Occurrence of Holes and Clashes: The top chart witnesses that the average number of holes per team per match (secondary ordinate) has diminished substantially over the years. More relevantly, both charts highlight the development of clashes (black holes): The median and average number of clashes per team per match has seen a strong decline in RoboCup's first decade. However, the problem of their occurrence has never fully disappeared. Using a log scale ordinate, the bottom chart puts an emphasis on the more recent 15 years and also includes data on games that were maximally biased by clashes (after having removed outliers).

for the median team. Additionally, we have plotted the the maximum number of clashes of the non-outlying teams over all matches – a value that can be used to confirm or reject the conclusion that competition results were, at least in part, biased by clashes. So, for example teams YuShan'22 and FRA-UNIted'22 had to face 4.6 and 4.2 clashes per match on average, but there were matches with as many as 39 or 32 clashes.

The 2022 RoboCup tournament stands out insofar as two of the considered measures, the average number of clashes per team per match, as well as the maximum amount of clashes in a game (outliers removed in either case), obtain an all time high in that year for the period of the past 15 years. Since also the median stands out within the previous six years, we will return to this point in a later section.

4 Interdependency of Clashes and Team Performance

While our discussion has so far concentrated only on the pure existence of holes and clashes, their impact on an ongoing match has been disregarded. Our example from the Introduction, however, illustrates the fact that there is some – in all likelihood negative – influence on the playing performance of a team. In what follows, we aim at quantifying that effect.

4.1 Relation Between Clashes and Agent CPU Utilization

Assuming a stable network connection and a constant access to compute power, the occurrence of clashes is essentially subject to the amount of computations made by an agent when determining its next action ("think" in Fig. 2). Each soccer simulation team applies very different algorithms and, henceforth, consumes a different amount of CPU power in each time step. If for some agent it holds $\delta = o + t + a > 100ms$, where o is the time a see message sent by the Soccer Server needs to arrive at the agent, a is the time an act message sent by an agent needs to arrive at the server, and t the agent's "thinking" time, then a hole arises and a clash in the subsequent cycle becomes likely.

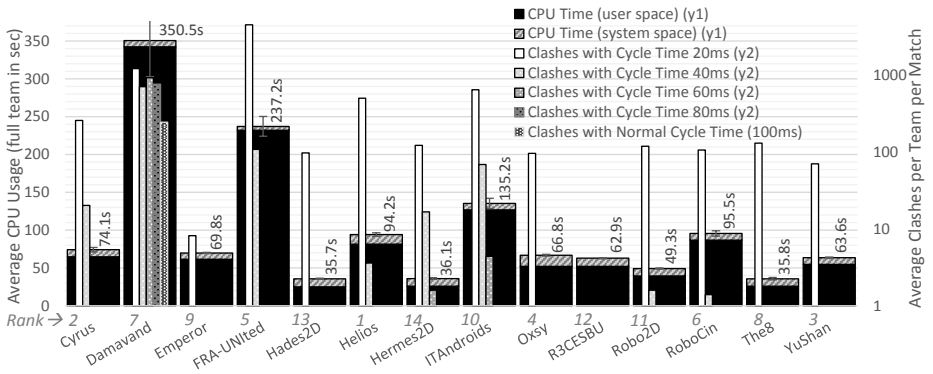

Fig. 4. CPU Utilization vs. Forced Clashes: The average summed CPU usage of an entire team during one match varies strongly across the 2023 SS2D teams (wide bars). We can easily provoke the emergence of clashes by artificially speeding up the timing of the Soccer Server, i.e. reducing the length of a single time step to $t_{step} < t_{default}$; the average number of clashes per team per match for $t_{step} \in \{20, \ldots, 100\}$ are plotted using narrow bars on the log-scale secondary y-axis.

Focusing on all teams participating in last year's world championships (2023), we determined the average required CPU utilization by the agents. Fig. 4 shows the average summed CPU time that the ensemble of the processes of a team spent during a single match in total (default duration of a match is 600s), executing code in user space (time spent by the processes executing their own code) plus the seconds executing code in system space, particularly the time spent by the Linux kernel on behalf of the process, such as handling system calls or executing kernel-level tasks[1]. The reported CPU times depend, of course, on the hardware and were produced by the teams playing on an x86_64 Intel Core i7-1165G7 CPU with 8 cores at 2.8GHz with the Soccer Server's default timing of $t_{default} = 100ms$.

[1] As an interesting side note, we found that more compute does not necessarily imply a better ranking in the tournament, with both variables being only slightly negatively correlated (Pearson correlation of -0.228).

4.2 Modifications to the Soccer Server's Timing

Speeding up the simulation (tested up to a factor of five), we can easily provoke clashes since agents will frequently fail to ensure $\delta \leq t_{step}$. As expected, there is a strong correlation between the agent's general demand for compute power and the number of clashes they produce when faced with a sped-up simulation. The narrow bars in Fig. 4 show that 12 out of the 14 teams considered generate more than 100 clashes per team per match for $t_{step} = 20ms$, with this number dropping when t_{step} is increased, until for $t_{step} \geq 80$ no more clashes are observed (except for the already discussed outlier Damavand) on the hardware mentioned. Note that in this setting the machine's eight cores are shared across both teams, granting (with the mentioned exception) sufficient computational resources to any of the 25 processes involved in a match (2x11 players, 2 coaches, 1 server).

4.3 Limitations of Computational Resources

Another way to provoke the appearance of holes and clashes, is to artificially restrain the computational resources granted to the agents while sticking with the default timing. This can be achieved when running the mentioned 25 processes within Docker [5] containers each, as it is regularly done during RoboCup world championships since 2022 [4]. So, while we considered "bare metal" matches in the preceding sections, we now focus on Docker-based settings. In addition to that, we abandon the inspection of a full age group of teams, but focus on a single pair made up of our team FRA-UNIted and 2023's champion Helios [2].

4.3.1 Limiting the Number of Cores per Teams

Using Docker, it is straightforward to specify on start-up which CPU cores can be utilized by which team. In preliminary experiments we verified, that zero holes and clashes occur for the considered match-up when both can utilize four or more cores. Using three cores, Helios never produces a clash, while FRA-UNIted starts doing so due to its higher computational demand (cf. Fig. 4).

Fig. 5 summarizes what happens, if Helios remains to be allowed to exploit three cores, while FRA-UNIted's ensemble of 12 agents must share three, two, or only one core(s). As to be expected, the sheer amount of clashes increases with the restriction of computational resources. Using three cores we observe, that there is no clash at all in 92% of the matches, while in the remaining 8% of the matches only 1-5 clashes occur. By contrast, sharing a single core only, merely a quarter of the matches remains clash-free, while in 40% of the matches more than 10 clashes occur. The middle part of the figure visualizes, how limiting the number of cores for one competitor (here: FRA-UNIted) affects its overall performance: When the number of cores is decreased, the amount of clashes occurring increases and simultaneously that team receives more goals against. This relation is made more transparent in the right part of the figure where matches with a certain number of clashes are grouped and average match results for those groups are plotted. Starting from ten clashes per match, a clear deterioration of the overall team performance must be acknowledged.

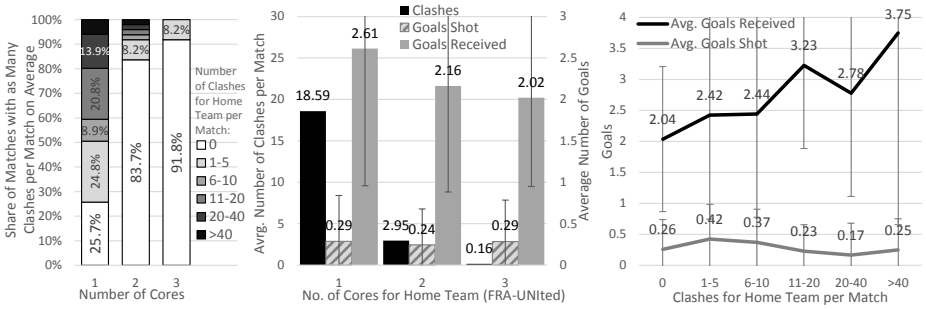

Fig. 5. Focusing on the fixture of the 2023 binary of FRA-UNIted (rank 5, dubbed home team) and champion Helios, the impact of reducing the available CPU cores for one team only (home team) is shown: Less than 3 cores imply more matches with a larger share of clashes (left) and an increased number of goals received (middle). Right: The average score of a match depends on the amount of clashes the home team had.

4.3.2 Constraining Core Usage of an Agent

We performed a more fine-grained analysis of the impact of clashes on a team's playing strength by instructing the Docker containers that run a player process to consume only a *fraction* of a CPU core's computing power. Initial tests have shown that a CPU core share of 0.3 per agent yields zero holes/clashes for neither FRA-UNIted nor Helios on the mentioned hardware, which is fully in line with our previous observations since 11 players plus the coach would make up for up to 3.6 utilized cores per team.

In a series of test games, we varied the CPU power granted to either team's agents within $I = [0.01, 0.3]$ cores per agent and measured the number of holes and clashes as well as average scores. Fig. 6's abscissa (left) lists all CPU share settings examined in the format as $x : y$ where $x \in I$ is the CPU share granted to any agent of the home team (FRA-UNIted) and $y \in I$ the share granted to any away team agent (Helios). The undistorted clash-free default setting is highlighted (gray background), left of which we have scenarios that thwarted the home team and right of which scenarios that impede the away team.

Besides the expectable insight that holes and clashes do apparently compromise a team's performance in terms of goals scored/received, it is striking to realize that as little as about ten clashes per team per match yield a significant change of the average score and, moreover, that as little as approximately 70 clashes ($0.30 : 0.05$ setting) depose last year's the champion team making it play inferior to that year's rank 5 team. We emphasize that all the clash values we report refer to the overall number of clashes for the *entire team* per match (not per agent per match).

The right part of Fig. 6 stresses the relation between clashes and performance by visualizing the same data differently. The x-axis denotes the share of home and away team CPU usage shares – a value of 0.5 indicates a fair sharing of resources whereas a value of 0.2 means that the home team was granted only 20% of the

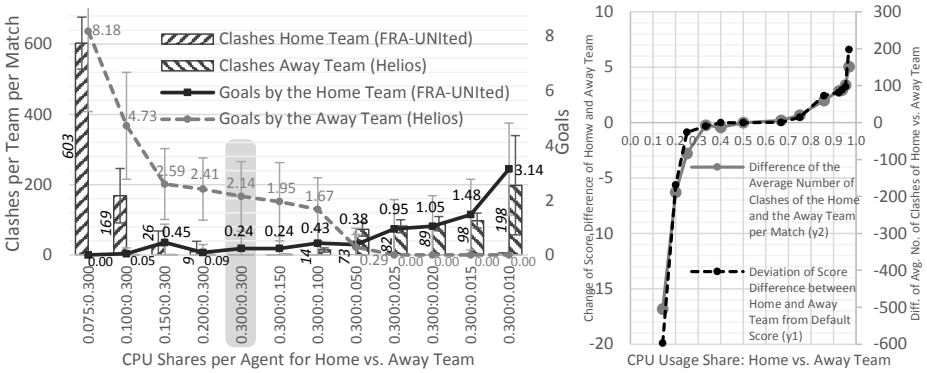

Fig. 6. The left chart shows how the balance of powers between the two considered teams changes w.r.t. clashes which we provoke by constraining the CPU core usage of a team's agents. The right chart takes an equal usage of compute power ($0.5 = \frac{0.3}{0.3+0.3}$) as baseline and opposes the difference of clashes (y_2) for both teams with changes in the score difference (y_1), highlighting the strong correlation between both.

overall computing power and the away team took 80%. The gray/black curves show the variations relative to the default setting ($0.3 : 0.3 \mathrel{\widehat{=}} 0.5$) in terms of for which team clashes occur as well as how, thus, the average score difference between home and away team changes. We observe a strong correlation between clashes and score difference shift (Pearson correlation of 0.74).

4.4 Competition-Like Set-Ups

Given that small amounts of clashes can distort soccer simulation games substantially, care must be taken with competition set-ups. To this end, the local organizers of the RoboCup 2023 soccer simulation 2D tournament did an excellent job by performing exhaustive analyses and tests in advance which are precisely documented [3] and which made the tournament a clash-free one.

This section addresses one factor that we judge to be a catalyst for the frequent re-occurrence of clashes in the preceding year's competition (2022, cf. Fig. 3), where we observed 10 matches with ≥ 10 clashes per team per match (maximum: 39). As a matter of fact, since the number of cores per team was 12 in that tournament and considering our findings from Section 4.1, it is obvious that, actually, no clashes should have occurred at all.

For an accurate replication of a competition site, we used a CPU cluster of 8x Dell PowerEdge R7525 with 32 cores at 3.0GHz and executed randomly selected Docker-based matches using the officially released 2023 binaries using 25 full cores per match in Ubuntu 22.04. Alongside, however, we started and stopped random Docker-based matches whose containers were run on the remaining cores. We were not interested in the results or clashes of those secondary matches, but in the question whether the existence of a parallel match exerts some interference with the run of the primary one. Unfortunately, we had to

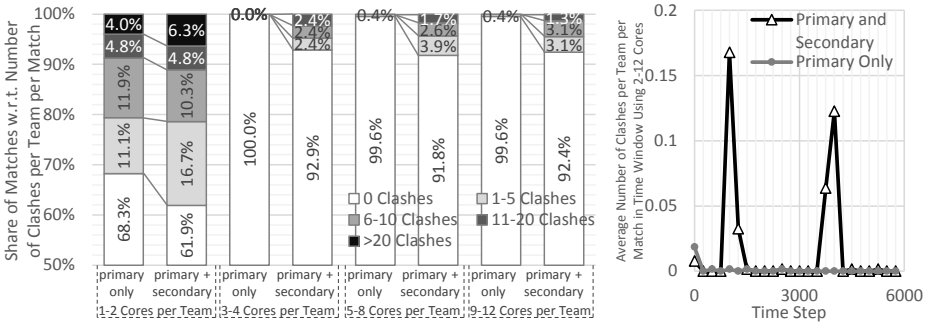

Fig. 7. Left: Running a single (primary) or two matches (primary and secondary) simultaneously on the same CPU cluster: We plot the share of clashes during primary matches (0, 1-5, 6-10, 11-20, or more than 20 clashes per team per match) and compare what happens, if that match runs solitarily vs. when some secondary match is started and run in separated Docker containers on the same CPU cluster. Apparently, more and more games become affected by clashes to varying degrees. Right: Throughout our experiments, two secondary matches of 500 time steps length were started. A first one within the first half of the primary match, the second one in the second half. The histogram shows the average number of clashes per team per match using a window size of 250 and reveals a clear link between clashes and the start-up phase of the secondary match. Note that this histogram covers only matches with three or more cores per team such that the lack of compute power as a cause for clashes is filtered out.

corroborate that conjecture as Fig. 7 shows. While expectable when the primary match was distributed on a few cores only, we observe the emergence of a critical amount of clashes per team per match even when the computational load of the primary match is spread across a large number of cores (e.g. 9-12), as soon as a secondary match takes place. The right part of Fig. 7 reveals the cause. The start-up of a Docker-based match takes about 45s and includes the launch of 25 Docker containers. We timed our experiments such that the primary match was superimposed with two such secondary matches (each one lasting 500 time steps) of which the first was launched in the first half and the second one in the second half time of the primary match. The corresponding peaks around $t = 1000$ and $t = 4000$ confirm the interference resulting from the fact that the operating system is in many instances occupied largely by the simultaneous start-up of the secondary match.

Inspecting the exact game starting times and time required for a Docker-based match start-up we could confirm that the clash that resulted in a missed tackling (cf. Fig. 1) took place exactly when on the same machine, despite using 25 separate CPU cores, the game between RoboCIn and HfutEngine was started simultaneously and, therefore, was very likely the result of the problems delineated in this section. Therefore, we conclude that running two or even more Docker-based soccer simulation 2D matches on a single machine, even when equipped with an abundance of cores, might distort games and ought to be disregarded or at least safeguarded by extensive tests in advance.

5 Conclusion

Establishing a reliable synchronization in real-time multi-agent systems is a delicate task. The Soccer Simulation 2D community as a whole met this challenge already in the late 1990s and thought it to be solved, with holes and clashes as main manifestations of mis-synchronization remaining as a side issue nowadays. The results of the analyses we presented in this paper show which significance the topic has had historically in the preceding 25 years of soccer simulation. In particular, our experiments revealed that, over the past few years, the topic has witnessed a clear resurgence in relevance.

Acknowledgements: This research has been supported by the German Federal Ministry of Education and Research under grant number 13FH027Kl1.

References

1. The RoboCup Online Archives of Soccer Simulation 2D Binaries and Logfiles (Last accessed: 06/2024), http://archive.robocup.info/Soccer/Simulation/2D/, https://chaosscripting.net/files/competitions/RoboCup/
2. Akiyama, H., Nakashima, T., Hatakeyama, K., Fujikawa, T., Hishiki, A.: HELIOS2023: RoboCup 2023 Soccer Simulation 2D Competition Champion. In: RoboCup 2023: Robot World Cup XXVI, to appear. Springer (2024)
3. Amini, O., Rad, A.: Unraveling the Mysteries of the Hole and Clash (Black Hole) Problem (Last accessed: 06/2024), https://www.rcss.dev/holesclashes
4. Amini, O., Rad, A., Zare, N.: SS2D Docker Tournament Runner (Last accessed: 06/2024), https://github.com/RCSS-IR/SS2D-Docker-Tournament-Runner
5. Boettiger, C.: An Introduction to Docker for Reproducible Research. ACM SIGOPS Operating Systems Review 49(1), 71–79 (2015)
6. Breunig, M., Kriegel, H., Ng, R., Sander, J.: LOF: Identifying Density-Based Local Outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. pp. 93–104. ACM, Dallas, USA (2000)
7. Butler, M., Prokopenko, M., Howard, T.: Flexible Synchronisation in RoboCup Environment: A Comparative Analysis. In: RoboCup 2000. pp. 119–128 (2001)
8. Cavalcanti, L., Joaquim, R., Barros, E.: Optimized Wireless Control and Telemetry Network for Soccer Robots. In: RoboCup 2021. pp. 177–188. Springer (2022)
9. Gabel, T.: The RoboCup-sim Mailing List Archive (Last accessed: 02/2024), https://lists.robocup.org/archives/list/robocup-sim@lists.robocup.org/2023/5/
10. Levesque, M., Tipper, D.: A Survey of Clock Synchronization Over Packet-Switched Networks. IEEE Comm. Surveys and Tutorials 18(4), 2926–2947 (2016)
11. Montgomery, J., Mackworth, A.: Adaptive Synchronisation for a RoboCup Agent. In: RoboCup 2002: Robot World Cup VI. pp. 135–149. Springer (2003)
12. Noda, I., Matsubara, H., Hiraki, K., Frank, I.: Soccer Server: A Tool for Research on Multi-Agent Systems. Applied Artificial Intelligence 12(2-3), 233–250 (1998)
13. Oliveira, L., Almeida, L., Santos, F.: A Loose Synchronisation Protocol for Managing RF Ranging in Mobile Ad-Hoc Networks. In: RoboCup 2011: Robot Soccer World Cup XV. pp. 574–585. Springer, Istanbul, Turkey (2011)
14. Polani, D.: Towards a Probabilistic Asynchronous Linear Control Theory. In: RoboCup 2003: Robot World Cup VII. pp. 720–728. Springer (2004)
15. Prokopenko, M., Wang, P.: Disruptive Innovations in RoboCup 2D Soccer Simulation League. In: RoboCup 2016: World Cup XX. pp. 529–541. Springer (2017)