Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

AUTONOMIC ORCHESTRATION OF IN-SITU AND IN-TRANSIT DATA ANALYTICS FOR SIMULATION STUDIES

Xiaorui Du
Zhuoxiao Meng
Anibal Siguenza-Torres
Alois Knoll

Technical University of Munich
Arcisstraße 21
Munich, 80333, GERMANY

Adriano Pimpini

Sapienza, University of Rome
Via Ariosto 25
Rome, 00185, ITALY

Andrea Piccione
Stefano Bortoli

Huawei Munich Research Center
Riesstraße 25
Munich, 80992, GERMANY

Alessandro Pellegrini

University of Rome Tor Vergata
Viale del Politecnico 1
Rome, 00133, ITALY

ABSTRACT

Modern parallel/distributed simulations can produce large amounts of data. The historical approach of performing analyses at the end of the simulation is unlikely to cope with modern, extremely large-scale analytics jobs. Indeed, the I/O subsystem can quickly become the global bottleneck. Similarly, processing on-the-fly the data produced by simulations can significantly impair the performance in terms of computational capacity and network load. We present a methodology and reference architecture for constructing an autonomic control system to determine at runtime the best placement for data processing (on simulation nodes or a set of external nodes). This allows for a good tradeoff between the load on the simulation's critical path and the data communication system. Our preliminary experimentation shows that autonomic orchestration is crucial to improve the global performance of a data analysis system, especially when the simulation node's rate of data production varies during simulation.

1 INTRODUCTION

Simulation is essential in many application areas, establishing itself as a third way of science alongside theory and on-field experimentation. Simulation studies constitute increasingly sophisticated processes involving large-scale models and numerous simulation experiments, often computationally intensive. To support the execution of these models, relying on highly parallel and distributed High-Performance Computing (HPC) systems is imperative. HPC simulations enable unprecedented levels of accuracy in models and make previously intractable problems tractable by exploiting distributed memory. They can allow for practical what-if analysis, as alternative scenarios can be explored through multiple concurrent simulations. However, the great detail achieved in simulations poses a significant problem in analysing simulation results.

From the point of view of computing resources, we can now exploit exascale architectures (Dongarra et al. 2019) thanks to extraordinary worldwide results (Kothe et al. 2019; Gagliardi et al. 2019). Anyhow, it is evident that managing data input/output (I/O) is one of the main bottlenecks compared to previous petascale systems (Lang et al. 2009). To overcome this limitation, many techniques have been proposed on the hardware side (see, e.g., Liu et al. 2003; Liu et al. 2012), whose ultimate aim is to reduce the impact of buffering or manipulation by software to move data.

However, these performance improvements are orthogonal to the fundamental design challenges of deciding how to distribute the simulation data and where to process it. As shown by Pu et al. (2015), the classical approach of aggregating all data in a single node significantly inflates the timeliness of analyses. In this direction, various techniques have been proposed to improve the performance of data analytics activities. For instance, Ellis (2014) showed that stream-processing-based data analysis is promising since it allows processing or generating even visual representations in real-time. However, it is challenging because the speed of receiving data and the limited time to understand it make it difficult for analysts to sufficiently examine the data without missing significant changes or patterns (Kesavan et al. 2020).

To reduce the load on the network, *in-situ* workflows have become popular (Larsen et al. 2017; Childs et al. 2022). They integrate the processing of data generated in the simulation pipeline to transfer already aggregated or pre-processed data to the analytics nodes. Anyhow, this strategy may lead to performance disadvantages because the processing tasks are placed on the simulation critical path. With this in mind, Bennett et al. (2012) showed how, for some simulation models, it is possible to construct an efficient organization in which part of the data processing takes place in situ and part on remote nodes (called *in-transit* processing). They divided in-situ and in-transit processing statically by analyzing the model's requirements. Identifying an optimal division can be a complex task, in general: given a set of queries on the output of a simulation, determining which part of the analysis must be performed on the simulation nodes and which on the external nodes is a problem that depends directly on the queries performed (e.g., if aggregate metrics are of interest, it is imperative to transfer all or part of the data).

Similarly, the dynamics of the model also play a key role. If a model is not balanced, different simulation nodes may observe a diverse workload in the in-situ processing activities, even for the same queries. This phenomenon, related to the need for synchronization to ensure consistency, can lead to disappointing performance results, regardless of whether the simulation synchronization scheme is conservative (e.g., Chandy and Misra 1979) or optimistic (e.g., Jefferson 1985). The performance drop can be even greater if the amount of data generated by the simulation varies over time due to highly-dynamic scenarios.

We address the issues above by studying an *autonomic self-optimization technique* (Kephart and Chess 2003) to orchestrate simulation data analytics. Through our approach, we show how it is possible to decide which portion of data processing should be performed in transit or in situ, depending on the runtime dynamics of the simulation model and the particular type of data analytics queries. We emphasize that the goal of this paper is not to propose an optimal self-optimization technique but rather to highlight the need for such approaches to deliver competitive simulation-analytics systems. Additionally, we depict a distributed architecture for simulation and data analytics that enables us to execute diverse user-defined queries competitively, leveraging our autonomic methodology. This architecture is archetypal of distributed data analytics jobs and is therefore organized to decouple data generation from the data processing part. The autonomic orchestration is based on live data obtained from software probes injected into the various components and on historical data that reduce the exploration time of the different configurations.

We exercise our methodology against multiple configurations of an agent-based simulation model executed on top of the CityMoS simulation framework (Zehe et al. 2017), a high-performance microscopic traffic simulator. The choice of microscopic traffic simulation as a subject of this study was made because it can provide large amounts of data to support multiple data analytics queries. Overall, our experimental evaluation shows that self-adaptive orchestration of the data analytics activities can deliver a non-negligible reduction in the time-to-completion of the joint simulation and analytics activities.

The remainder of this paper is structured as follows. In Section 2, we formalize our problem statement, illustrating the motivations of our work. Section 3 discusses related work. The autonomic orchestration methodology is described in Section 4. An experimental assessment is presented in Section 5.

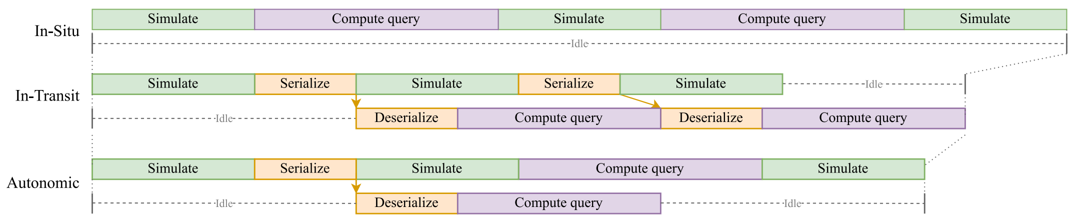


Figure 1: Variation of end-to-end completion time when using in-situ, in-transit, or autonomic computation.

2 BACKGROUND AND MOTIVATION

Considering a generic multi-query analytics study based on n simulation and m processing nodes, deciding upon in-situ or in-transit data processing can be based on the notion of *end-to-end completion time*. This notion captures the idea that the user is typically interested in obtaining the result of the computation of the data independently of how the data processing is executed. Therefore, it is reasonable to try to reduce the time-to-solution, which may behave differently depending on the processing paradigm used.

Figure 1 depicts from a high level the possible dynamics of in-situ or in-transit data processing. In the former case, simulation nodes may dedicate non-negligible time to data processing, giving the following formulation of the the end-to-end completion time:

$$T_{situ} = \max_{i\in[1,n]} \left(t_{sim_i} + t_{proc_i}\right), \tag{1}$$

where t_{sim_i} represents the time required to run the simulation on the i-th simulation node, while t_{proc_i} accounts for the time needed to collect the relevant data from the simulation state and process the queries. Conversely, in the second case, the time spent on data serialization/deserialization can greatly lengthen the time needed to obtain the final results. In this case, the end-to-end completion time can be captured as:

$$T_{transit} = \max_{i\in[1,n]} \left(t_{sim_i} + t_{ser_i} + t_{idle_i}\right), \tag{2}$$

where t_{ser_i} captures the time to serialize the portion of the simulation state used for query evaluation, and t_{idle_i} accounts for the time to complete the query processing at the end of the simulation. Despite high-level, Equations (1) and (2) provide an insight into the effect of runtime dynamics on end-to-end computation. If the amount of data generated by the simulation is large, then t_{ser_i} grows. At the same time, in the case of in-transit processing, simulation nodes could be idle for a long time: t_{idle_i} can be a dominating factor.

The effects of Equations (1) and (2) are shown empirically in Figures 2 and 3, where we report the simulation results of a synthetic grid road network composed of 100 intersections, 81 traffic lights, and 800 lanes with a peak of $\sim$2,500 agents. The total length of the road network is 41 km. Although simple, this model has a time-varying workload associated with the number of active vehicles—the bell curve in the figures. From the in-situ results (Figure 2), we observe that processing activities become dominant when the number of vehicles is high—t_{proc_i} increases. Conversely, in the in-transit case (Figure 3), the waiting time to process a single data point rises as the analytics nodes start lagging behind.

Even in this simple model, it is not possible to know a priori whether $T_{situ} > T_{transit}$ since this depends on the dynamics of the model and the involved queries. The best scenario for minimizing end-to-end completion time is obtained when the coupled simulator-analytics system can determine timely whether, at a certain execution stage, it is more convenient to transfer data to the remote nodes for analysis or it is more convenient to carry out (partial or full) processing directly on the simulation nodes.

To support this decision, we propose the inclusion of an autonomic orchestrator that jointly monitors the progress of simulation and data analysis. It is capable of interacting with the query processing activities to migrate the computation towards the hybrid in-situ/in-transit configuration that can reduce the completion time of the analytics job. As we will show, even with the use of a simple autonomic model, this approach can significantly reduce the end-to-end time even for complex simulation scenarios, allowing tasks to be orchestrated as shown in Figure 1.

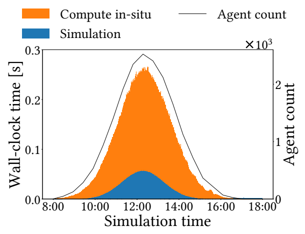


Figure 2: In-situ computation characterization.

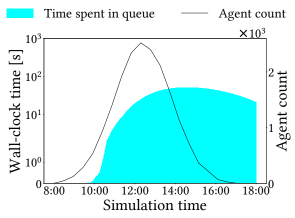


Figure 3: Buffering time for in-transit computation.

3 RELATED WORK

The performance gap between parallel/distributed simulations and I/O systems has expanded the number of works dealing with in-situ or in-transit processing. In the case of in-situ simulations, many works propose approaches that attempt to reduce the completion time of data analysis by resorting to dedicated hardware (e.g., Li et al. 2010; Zhang et al. 2012) or dedicated software solutions (e.g., Zheng et al. 2010; Vishwanath et al. 2011; Chen et al. 2016). Conversely, in the context of in-transit simulations, various combinations of data staging areas have been studied (see, for example, Abbasi et al. 2009; Zheng et al. 2010; Vishwanath et al. 2011). Godoy et al. (2020) attempt to provide a unified API to support the movement of data generated by HPC systems, also exploiting cloud environments.

Both of these approaches aim to improve performance: the former removes some of the data analysis work from the critical path of the simulation, while the latter seeks to exploit asynchronous data movement to reduce the impact of I/O operations. Our approach is orthogonal in that it attempts to adaptively combine both methodologies, using runtime performance data to make a choice that reduces the total time to complete a joint simulation-analysis job.

Isaacs et al. (2014), Muelder et al. (2016), Sanderson et al. (2018), Kesavan et al. (2020) have considered real-time visualization of data from highly parallel applications or systems as a reference scenario for on-the-fly data processing. These solutions are tailored for computationally intensive tasks such as data visualization, while our goal is to support generic data processing, even supporting various queries at the same time. Bennett et al. (2012) start from considerations similar to those underlying this work, reasoning about the opportunity to combine in-situ and in-transit processing to improve the performance of data analytics approaches. Unlike Bennett et al. (2012), our work considers the possibility of determining at runtime which is the best way to evaluate queries on the data produced by the simulation to maximize performance even in the presence of highly dynamic workloads.

The importance of combining in-situ and in-transit processing is also discussed by Zheng et al. (2011), providing a performance model to quantitatively describe the trade-offs of these two approaches in a simulation. However, this model is based on the total cost of queries after simulation, limiting its applicability in highly dynamic scenarios. Furthermore, the model assumes that processing the data generated by the simulation is faster than the simulation itself. This assumption may fall if the queries to be evaluated are large in number or very complex.

Similarly, Zou et al. (2014) explore in-transit and in-situ selection strategies by analyzing the possibility of reducing data movement and eliminating severe I/O performance bottlenecks. The proposed algorithm allows for runtime selection; however, in order to work, it requires idle nodes to be started and a (baseline) subset of data to be processed to estimate the cost of the upcoming query, which could impose significant overhead on the simulation.

The idea of moving the computation instead of the data was explored by Docan et al. (2011). They share with us the concept of performing the same computation on the data in different locations, depending on performance requirements or the load on the I/O system. However, our work introduces the possibility of *partial* computations, making it possible to compute specific metrics in situ and others in transit, depending on the overall load on the system. Furthermore, our goal is not to migrate the executable that performs the computation from one node to another since queries of interest to data analysts are easily distributed across all nodes beforehand.

4 AUTONOMIC DATA ANALYTICS ORCHESTRATION

In the definition of our policy, we assume a finite set of metrics denoted as $m_0, m_1, \ldots, m_n$, and we consider a scenario where the system user may desire to evaluate these metrics at arbitrary time points t during the simulation. Such computations are referred to as *queries*. The time required to evaluate a query q depends on a specific data set collected from the simulation state to evaluate the metrics associated with q. We call it the query's *content*. Specifically, we are interested in determining the size of the content, denoted as $size(q)$. We assume that the simulation and the analytics processes are independent, possibly running on their own machines connected through a network.

4.1 The Autonomic Orchestration Algorithm

Let us start analyzing the cost of a single query q. From the perspective of one simulation node s, we can model the cost of evaluating q as follows:

$$t_{situ_s} = [execute_s(q)]_t \tag{3}$$

$$t_{transit_s} = [serialize(q) + send(q)]_t \tag{4}$$

At first glance, we should always greedily choose the approach with the lowest estimated cost between the two. This would minimize the end-to-end time on the simulation node, and, in the case of very lightweight analytics, it could also effectively solve the problem.

In a realistic scenario, this choice would be biased towards the in-transit strategy; for non-trivial queries, the local execution cost would likely be higher than serialization and network activities, also because they could be partially carried out in the background. The large amount of data processed in transit would result in a growing number of tasks waiting for processing in the analytics node. Ignoring memory limitations, these tasks would continue being processed long after the simulation node has completed its computation. Therefore it is necessary to take into account the costs of both strategies also on one analytics node a:

$$t_{situ_a} = 0 \tag{5}$$

$$t_{transit_a} = [receive(q) + deserialize(q) + execute_a(q)]_t \tag{6}$$

Highly-dynamic workloads and possible network delays make it hard to determine these costs precisely when making a decision; thus, we must rely on some simplifying assumptions. We use a simple linear approximation, assuming that each query m_k has a relatively-stable cost per byte, observable during the system's lifetime. We can then derive the per-byte cost S_k for the in-situ computation on simulation nodes, and the per-byte costs T_k and A_k for the in-transit strategy on the simulation and analytics nodes respectively.

Deciding how to merge the two cost estimates to make a good decision can be tricky, especially when multiple simulation and analytics nodes are used. Since the system can comprise any number of independent nodes with overlapping computations in wall-clock time, it is unfeasible to satisfactorily estimate or characterize this global end-to-end time, particularly given the unknown and variable simulation computational load.

Thus, our autonomic algorithm operates under a "best-effort" approach considering two key factors.

1. If an analytics node is experiencing a high computational load, new in-transit computations may have to wait in the queue for a long time. If the simulation nodes' computation were to complete soon, waiting for the analytics nodes would be undesirable.
2. Since the completion of any subsequent activity depends on the simulation computation, the default optimal approach is to minimize the computational cost on the simulation nodes, if possible.

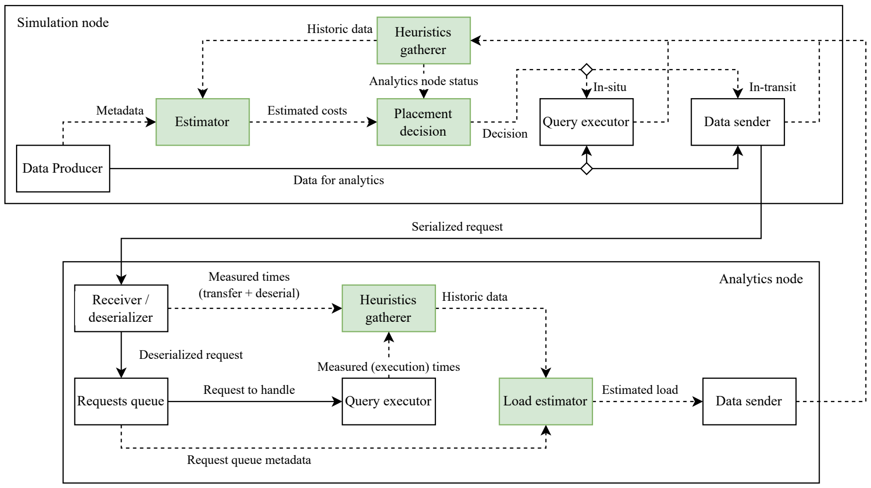


Figure 4: The distributed architecture for autonomic orchestration of data analytics jobs.

Therefore, our primary objective is to utilize the greedy approach as much as possible while ensuring the analytics nodes are not overloaded. We introduce a new time quantity, t_{wait_a}, defined as the duration for which a new query request must wait before being processed by the analytics node a. Although the exact value of t_{wait_a} is not known at all times, an analytics node a can estimate it by combining the estimated $t_{transit_a}$ of the queries in its queue. With $type(q)$ denoting the type of the query q, we can estimate the waiting time using the following equation:

$$t_{wait_a} = \sum^{queue_a} size(q) \cdot A_{type(q)} \tag{7}$$

It should be noted that Equation (7) should not account for the network transfer time $receive(q)$ for queries that are already received by the analytics node a and are waiting in its queue to be executed. While a precise estimation would consider this distinction, it has been omitted here for brevity and simplicity.

Now, we can introduce a baseline autonomic decision policy as follows: first, we set a threshold time t_{max} as the maximum time the user may want to wait for the given analytics job after the simulation has been completed. Then, we evaluate t_{wait_a}; if it is greater than t_{max} for all analytics nodes, we execute the query in situ. Otherwise, we apply the simulation node's greedy decision criterion and randomly select one of the analytics such that $t_{wait_a} < t_{max}$. Assuming long simulation runs, t_{wait} and the other relevant quantities can be lazily updated periodically.

If, at any moment, a simulation node performs analytics computations in situ, it will allow the analytics node to catch up, reducing the estimated t_{wait} for future queries, which have a higher probability of being offloaded to the analytics node, maintaining dynamic equilibrium. Simulation node makes independent decisions on where the analytics processing should occur; thus there may be particular instants in which some analytics nodes may have a backlog of tasks such that $t_{wait_a} > t_{max}$. As we will show experimentally, even this simple approach can deliver non-negligible benefits in the overall time-to-solution. As mentioned, this paper's focus is not on identifying an optimal autonomic policy.

The time threshold t_{max} acts as a tolerance value, where a higher value increases the simulation node's resource utilization efficiency but may cause longer analytic node waiting times, which may be acceptable for long simulation runs. Conversely, a lower value of t_{max} guarantees that the analytic node will not get overloaded, resulting in sub-optimal use of the simulation node capabilities.

4.2 Reference Architecture

To deploy the aforementioned autonomic policy, we designed a logical architecture consisting of the previously-mentioned parts: a simulation and an analytics component. The main building blocks of this

Table 1: Analytics queries used in the assessment—n is the number of agents.

Query	**Description**
Agent count (Q1)	Read the pre-computed agent number per road. Computational complexity: $\Theta(1)$.
Average speed (Q2)	Average speed of agents. Computational complexity: $\Theta(n)$.
Top speed per road (Q3)	Get the speed of the top k fastest agents in each road. Computational complexity: $\Theta(n \log(n))$.
Agent distances (Q4)	Distance between all agents. Computational complexity: $\Theta(n^2)$.

architecture are shown in Figure 4. The architecture is general, so we have embedded multiple fundamental building blocks in it, although only the ones highlighted in green are relevant to our current proposal. As shown, the components establish a feedback loop. The autonomic orchestrator uses this loop to make decisions to balance computational load and data transmission costs. The simulation component sends data to the analytics component, which periodically responds by providing statistics on its internal load. These runtime measurements are used to cope with the non-constant simulation load and the varying network and system conditions.

Relying solely on the most recent measurement is insufficient, as sudden spikes in conditions can occur and significantly impact computation. This may lead to unstable and incorrect decisions made by the autonomic components of the system. The *Heuristics gatherer* component is designed to mitigate this issue by collecting measurements and aggregating them into more stable values—currently, we rely on an exponential moving average. In analytics nodes, the *Load estimator* uses local heuristics and pending requests' metadata to gauge the time to completion of all local jobs, as discussed in Section 4.1.

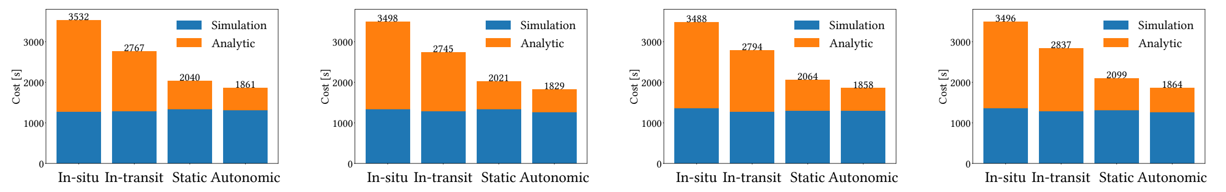
The actual autonomic decision-making combines the *Estimator* and the *Placement decision* components. The Estimator component utilizes metadata provided by the data producer (e.g., size and type of output data streams) and information about local and analytics nodes' processing performance to estimate the costs associated with both in-situ and in-transit processing strategies. The Placement decision component determines the appropriate processing strategy (in-situ or in-transit) for a given set of analytics, effectively serving as the core of the autonomic decision system. This is done by implementing the autonomic policy described in Section 4.1.

5 EXPERIMENTAL ASSESSMENT

Our simulation model consists of a road network and a series of itineraries that the cars follow. Specifically, the network we used for the experiments is the city of Shenzhen in China. The environment map was converted into a CityMoS (Zehe et al. 2017) configuration starting from OpenStreetMap data, using the tool by Meng et al. (2022). We have simulated 10 hours (an interval representative of a regular working day) starting at 8:00 am and ending at 6:00 pm.

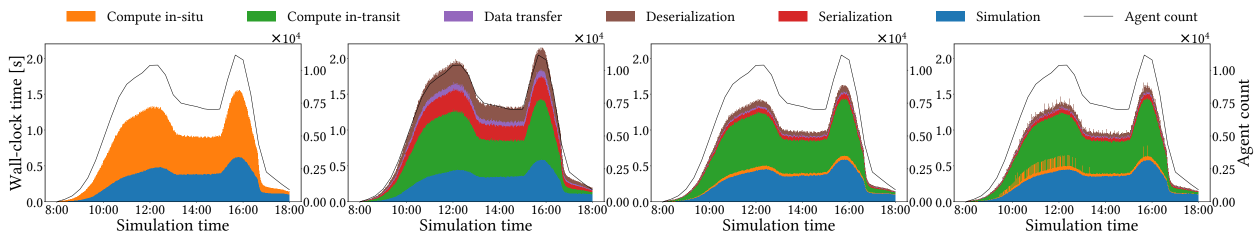
At simulation startup, the model is unloaded. The number of vehicles peaks at $\sim$10,600 at noon, slowly drops to $\sim$6,500 at 2:00 pm, and then regrows to $\sim$11,400 at 4:00 pm, then goes towards zero until the simulation ends. This oscillating behavior was explicitly configured to mimic a classical Gaussian Mixture Model of traffic volume over the day (Hu and Hellendoorn 2013). The vehicles follow itineraries generated based on origin-destination matrices between pairwise traffic analysis zones (Li et al. 2021). To emulate a complex analytics scenario, we have used a total set of 4 different queries evaluated over the entire simulation run. The queries we have used are reported in Table 1.

We have relied on a small cluster composed of 6 server machines, each one equipped with two Intel Xeon E5-2680 v3 @2.50 GHz CPUs with 12 physical/24 logical cores and 256 GB RAM each, connected via 10 Gbps Ethernet. We have conducted experiments using a variable number of simulation and analytics nodes using this cluster. We have also used a single node to host both the simulation and analytics components of Figure 4, to study the behavior of our approach in a severely constrained setup. In this scenario, we used the same number of simulation and data analytics cores in both setups to make the results comparable.



(a) 1-second resolution. (b) 4-second resolution. (c) 8-second resolution. (d) 16-second resolution.

Figure 5: Single node: end-to-end completion time with the same amount of processed data.



(a) In-situ processing (b) In-transit processing (c) Static assignment (d) Autonomic policy

Figure 6: Single node: workload composition across nodes, 8 seconds resolution.

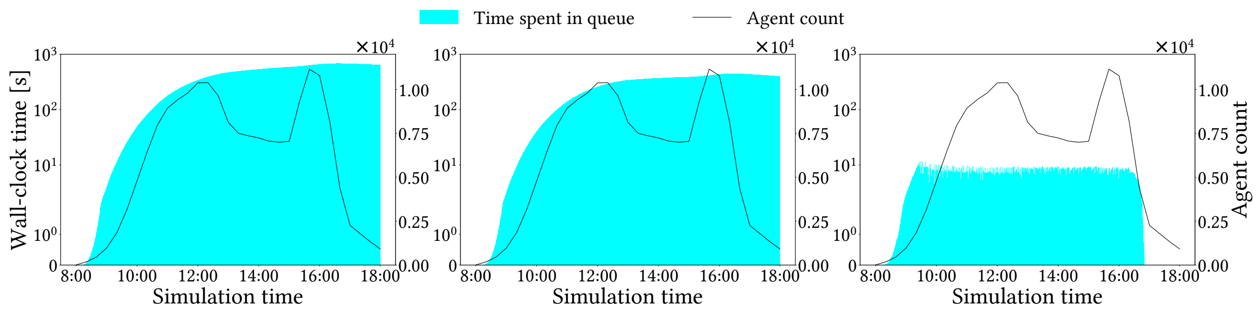
All the simulation runs have been configured using the same random seed, and the provided results are averaged over 5 different runs. We have also varied the *query evaluation resolution*, namely the amount of simulated time after which all queries are evaluated on the buffered data.

We have compared four different configurations to illustrate the benefits of the proposed autonomic approach. In the *in-situ* configuration, all analytics processing is executed on the simulation nodes on the simulation critical path. The *in-transit* configuration performs no computation on the simulation nodes, thus transferring all the data to the analytics nodes, in a round-robin fashion. The *static* assignment configuration is the optimal reference configuration. It is based on running all the simulation and analytics activities and determining post-mortem the best placement for the queries in Table 1, similarly to Zheng et al. (2011). After identifying the optimal configuration, we rerun the entire experiment to measure the end-to-end completion time. This approach allows evaluating, for every configuration, the best-suited processing configuration, determined by hand. Finally, the *autonomic* configuration evaluates the proposed autonomic orchestration approach.

5.1 Experimental Results

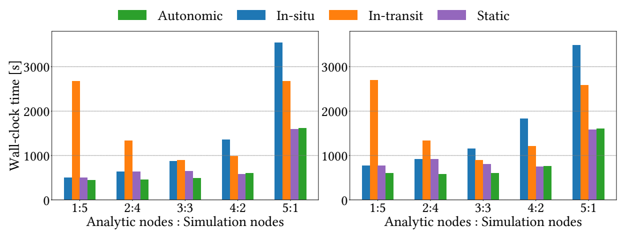
In Figure 5, we report the end-to-end completion times when the simulation and analytics components of Figure 4 are deployed on the same machine. As can be seen, the autonomic approach outperforms all configurations independently of the query evaluation resolution, providing a performance increment of up to 47%. The reasons for these results can be drawn from Figures 6 and 7, where we report the characterization of the simulation/analytics processing workloads and the time the data spent in the analytics node's queue before being processed, respectively. Similarly to Figure 3, we superimposed the model load over time. The plots show the values averaged every 48 seconds of simulated time, for visualization purposes.

As can be seen, in the case of in-situ processing, the simulation node spends a non-negligible amount of time computing the queries (Figure 6a). As an effect, the simulation throughput is reduced as the query evaluation happens on the critical path. An orthogonal scenario is observed when relying on pure in-transit processing (Figures 6b and 7a). Here, we observe that while the simulation node spends a non-minimal amount of time in serialization and data transfer activities, the analytics node's waiting time



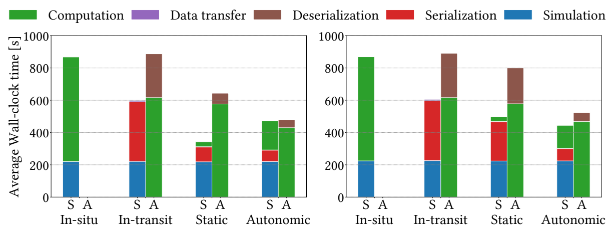
(a) In-transit processing (b) Static assignment (c) Autonomic policy

Figure 7: Single node: time the data spent waiting in the analytics node queue, 8 seconds resolution.



(a) Balanced workload (b) Imbalanced workload

Figure 8: Distributed setup with end-to-end time.



(a) Balanced workload (b) Imbalanced workload

Figure 9: Distributed setup with cost characterization.

grows unmanageable. Both phenomena are because the queries are evaluated over a large amount of data, especially when the workload peaks. Here, data buffered for data analytics increases significantly, delaying the end-to-end completion.

Interestingly, the static assignment configuration (Figures 6c and 7b) exhibits a comparable behavior. Although seemingly counterintuitive, the reason is in the variable workload. When the workload is low (e.g., at the beginning or end of the simulation), the time spent on serialization and data transfer activities on the simulation node (Figure 6c) is relatively low. Thus, the analytics node can quickly process the data received (Figure 7b). When the load increases, the processing time on the analytics node increases dramatically, accumulating delay. This delay is such that even when the load decreases (e.g., in the middle or at the end of the simulation), the analytics node relentlessly tries to catch up with the queued work, deferring the completion time.

The autonomic approach correctly captures the overloading of the analysis process and the increased workload on the critical path. Looking at the results in Figures 6d and 7c, we notice that the system continuously switches between in-situ and in-transit processing phases. This way, query evaluation activities are carried out simultaneously on both the simulation and analysis processes. The immediately observable effect is that, at the data analysis node (Figure 7c), the average data processing latency is kept around the tolerance threshold. At the same time, the proportion of time spent on simulation activities (Figure 6d) is greater, indicating a reduction in the load on the critical path.

In Figures 8 and 9, we show the behavior of the proposed approach in a distributed simulation setup. Here, we have varied the ratio of nodes used for simulation and analytics in a cluster of 6 machines. As can be seen, when the number of simulation nodes decreases, processing the queries in situ increases the completion time. This is expected because the total computing power available to process simulation and analytics activities is decreased. On the other hand, the in-transit configuration observes a reduced end-to-end time when increasing the number of nodes up to 3 analytics nodes and 3 simulation nodes.

Then, the end-to-end time rises again. This trend is explained by considering that when the number of simulation nodes is high (in the 1:5 ratio), the simulation speed is high because the distributed execution of the model can advance the simulation state very quickly. In this case, the analytics node is flooded with many queries, which prolongs the completion time—we observe the idle time depicted in Figure 1. The best ratio is found with an equal number of simulation and analytics nodes.

The autonomic orchestration approach can capture the different capabilities of the deployments. When the number of simulation nodes is reduced, it prefers the in-transit computation, allowing the simulation to keep up with the generation of new data. Conversely, when the number of analytics nodes is low, it effectively exploits in-situ computation to relieve the analytics node from the higher burden. Interestingly, the results for the distributed setup confirm our observations in Figure 5: the autonomic model either outperforms the optimal static configuration, thanks to the capability to continuously switch between in-situ and in-transit computation, depending on the current workload, or matches the static configuration performance. Again, this result is substantial because the autonomic model needs no apriori information to deliver a close-to-optimal result.

Moreover, in the configuration with an equal number of simulation and analytics nodes, namely when the in-situ and in-transit configurations show a comparable performance, we note that autonomic orchestration provides a large improvement over the static configuration. To better study the reasons behind this behavior, we provide in Figure 9a the characterization of the costs for this configuration. As can be seen, the autonomic approach correctly captures that the simulation node can also be exploited for analytics processing in certain phases of the execution. In that case, it switches to in-situ computation. This switch is done during the lifetime of the simulation for the different queries. Conversely, the static approach executes each query in situ or transit. The autonomic orchestrator is much more versatile and can respond promptly to execution dynamics.

The distributed setup we discussed distributes the road map evenly across the different simulation nodes. Given the length of the simulated timeframe and the agents' dynamics, the load on the various simulation nodes can be considered balanced. We have also studied the behavior of the autonomic approach in case of an imbalanced simulation—the corresponding results are provided in Figures 8b and 9b. In this configuration, the imbalance is generated by the different simulation nodes taking care of an uneven number of road segments (in the 5 simulation nodes case, each takes care of 10%, 15%, 20%, 25%, 30% respectively). As we can observe, the workload imbalance increases the time required to complete the simulation (except for the single simulation node case, where the single node keeps the same load). This is related to the increased idle time of the less-loaded simulation nodes every time that the distributed simulation has to synchronize. Despite this increment, the relative trends of the different configurations do not significantly change.

All these results are an indication of the versatility and resilience of the reference architecture proposed in this work since it can orchestrate data analysis activities in different deployments, even when non-minimal transmission delays are introduced–we recall that data analysis activities involve up to 300 GB of data potentially transferred between nodes.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented an autonomic orchestrator for the execution of data analysis jobs and a reference distributed architecture. Taken together, these two components make it possible to reduce the end-to-end completion time for evaluating multiple queries on data generated by simulations.

In future work, we plan to make the autonomic policy more resilient to sudden fluctuations in workload and to introduce additional dimensions to assist in the choice of the best placement of query computation. In particular, we want to investigate query optimization techniques that also consider the data of interest from the simulation state to avoid repeated access to the same parts of the state, e.g., to reduce serialization/ deserialization time.

REFERENCES

Abbasi, H., M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. 2009. “DataStager: Scalable Data Staging Services for Petascale Applications”. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, 39–48. New York, NY, USA: Association for Computing Machinery.

Bennett, J. C., H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. 2012. “Combining In-situ and In-transit Processing to Enable Extreme-scale Scientific Analysis”. In *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–9. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Chandy, K. M., and J. Misra. 1979. “Distributed Simulation: A Case Study in Design and Verification of Distributed Programs”. *IEEE Transactions on Software Engineering* SE-5(5):440–452.

Chen, C., M. Lang, L. Ionkov, and Y. Chen. 2016. “Active Burst-Buffer: In-Transit Processing Integrated into Hierarchical Storage”. In *Proceedings of the 2016 International Conference on Networking, Architecture and Storage*, 1–10. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Childs, H., J. C. Bennett, and C. Garth. (Eds.) 2022. *In Situ Visualization for Computational Science*. Mathematics and Visualization. Cham, Switzerland: Springer Nature.

Docan, C., M. Parashar, J. Cummings, and S. Klasky. 2011. “Moving the Code to the Data - Dynamic Code Deployment Using ActiveSpaces”. In *Proceedings of the 2011 International Parallel & Distributed Processing Symposium*, 758–769. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Dongarra, J., S. Gottlieb, and W. T. C. Kramer. 2019. “Race to Exascale”. *Computing in Science & Engineering* 21(1):4–5.

Ellis, B. 2014, June. *Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data*. Hoboken, NJ, USA: Wiley.

Gagliardi, F., M. Moreto, M. Olivieri, and M. Valero. 2019. “The International Race Towards Exascale in Europe”. *CCF Transactions on High Performance Computing* 1(1):3–13.

Godoy, W. F., N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, *et al*. 2020. “ADIOS 2: The Adaptable Input Output System. A Framework for High-Performance Data Management”. *SoftwareX* 12:100561.

Hu, Y., and J. Hellendoorn. 2013. “Daily Traffic Volume Modeling Based on Travel Behaviors”. In *Proceedings of the 10th International Conference on Networking, Sensing and Control*, 639–644. Piscataway, NJ, USA: IEEE.

Isaacs, K. E., P.-T. Bremer, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, and B. Hamann. 2014. “Combing the Communication Hairball: Visualizing Parallel Execution Traces using Logical Time”. *IEEE Transactions on Visualization and Computer Graphics* 20(12):2349–2358.

Jefferson, D. R. 1985. “Virtual Time”. *ACM Transactions on Programming Languages and Systems* 7(3):404–425.

Kephart, J. O., and D. M. Chess. 2003. “The Vision of Autonomic Computing”. *Computer* 36(1):41–50.

Kesavan, S. P., T. Fujiwara, J. K. Li, C. Ross, M. Mubarak, C. D. Carothers, R. B. Ross, and K.-L. Ma. 2020. “A Visual Analytics Framework for Reviewing Streaming Performance Data”. In *Proceedings of the 2020 IEEE Pacific Visualization Symposium*, 206–215. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Kothe, D., S. Lee, and I. Qualters. 2019. “Exascale Computing in the United States”. *Computing in Science & Engineering* 21(1).

Lang, S., P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. 2009, November. “I/O Performance Challenges at Leadership Scale”. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Number Article 40 in SC ’09, 1–12. New York, NY, USA: Association for Computing Machinery.

Larsen, M., J. Ahrens, U. Ayachit, E. Brugger, H. Childs, B. Geveci, and C. Harrison. 2017. “The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman”. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, 42–46. New York, NY, USA: Association for Computing Machinery.

Li, M., S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, *et al*. 2010. “Functional Partitioning to Optimize End-to-End Performance on Many-core Architectures”. In *Proceedings of the 2010 International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–12. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Li, Z., G. Xiong, Y. Zhang, M. Zheng, X. Dong, and Y. Lv. 2021. “Urban Trip Generation Forecasting Based on Gradient Boosting Algorithm”. In *Proceedings of the 1st International Conference on Digital Twins and Parallel Intelligence*, 50–53. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Liu, J., J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. 2003. “High Performance RDMA-based MPI Implementation over InfiniBand”. In *Proceedings of the 17th annual international conference on Supercomputing*, 295–304. New York, NY, USA: Association for Computing Machinery.

Liu, N., J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. 2012. “On the Role of Burst Buffers in Leadership-class Storage Systems”. In *Proceedings of the 28th Symposium on Mass Storage Systems and Technologies (MSST)*, 1–11. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Meng, Z., X. Du, P. Sottovia, D. Foroni, C. Axenie, A. Wieder, D. Eckhoff, S. Bortoli, A. Knoll, and C. Sommer. 2022. “Topology-Preserving Simplification of OpenStreetMap Network Data for Large-scale Simulation in SUMO”. In *Proceedings of the 2022 SUMO User Conference*, Volume 3, 181–197. Hannover, Germany: TIB Open Publishing.

Muelder, C., B. Zhu, W. Chen, H. Zhang, and K.-L. Ma. 2016. "Visual Analysis of Cloud Computing Performance Using Behavioral Lines". *IEEE Transactions on Visualization and Computer Graphics* 22(6):1694–1704.

Principe, M., A. Piccione, A. Pellegrini, and F. Quaglia. 2020. "Approximated Rollbacks". In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 23–33. New York, NY, USA: ACM.

Pu, Q., G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica. 2015. "Low Latency Geo-distributed Data Analytics". *SIGCOMM Comput. Commun. Rev.* 45(4):421–434.

Sanderson, A., A. Humphrey, J. Schmidt, and R. Sisneros. 2018. "Coupling the Uintah Framework and the VisIt Toolkit for Parallel In Situ Data Analysis and Visualization and Computational Steering". In *High Performance Computing*, edited by R. Yokota, M. Weiland, J. Shalf, and S. Alam, Volume 11203 of *LNCS*, 201–214. Cham, Switzerland: Springer.

Vishwanath, V., M. Hereld, and M. E. Papka. 2011. "Toward Simulation-time Data Analysis and I/O Acceleration on Leadership-class Systems". In *Proceedings of the Symposium on Large Data Analysis and Visualization*, 9–14. Piscataway, NJ. USA: Institute of Electrical and Electronics Engineers.

Zehe, D., S. Nair, A. Knoll, and D. Eckhoff. 2017. "Towards citymos: a coupled city-scale mobility simulation framework". In *Proceedings of the 5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication*, edited by A. Djanatliev, K.-S. Hielscher, and R. German, Volume CS-2017-03 of *Technical Reports*, 26–28. Nuremberg, Germany: Friedrich-Alexander-Universität Erlangen-Nürnberg.

Zhang, F., C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. 2012. "Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform". In *Proceedings of the 26th International Parallel and Distributed Processing Symposium*, 1352–1363. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Zheng, F., H. Abbasi, J. Cao, J. Dayal, K. Schwan, M. Wolf, S. Klasky, and N. Podhorszki. 2011. "In-situ I/O Processing: A Case for Location Flexibility". In *Proceedings of the Workshop on Parallel Data Storage*, 37–42. New York, USA: ACM.

Zheng, F., H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. 2010. "PreDatA – Preparatory Data Analytics on Peta-scale Machines". In *Proceedings of the 2010 International Symposium on Parallel & Distributed Processing*, 1–12. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers.

Zou, H., Y. Yu, W. Tang, and H.-W. M. Chen. 2014. "FlexAnalytics: A Flexible Data Analytics Framework for Big Data Applications with I/O Performance Improvement". *Big Data Research* 1:4–13.

AUTHOR BIOGRAPHIES

XIAORUI DU is a Ph.D. student at the Technical University of Munich (TUM) and Huawei Munich Research Center. His research interest is efficient data analytics for large-scale simulations. His email address is xiaorui.du@huawei.com.

ADRIANO PIMPINI is a Ph.D. student at Sapienza University of Rome. His research interests are high-performance computing and large-scale simulations, especially of Spiking Neural Networks. His email address is pimpini@diag.uniroma1.it.

ANDREA PICCIONE is a Senior Research Engineer at Huawei Munich Research Center. His research interests are high-performance computing and large-scale agent-based simulations. His email address is andrea.piccione@huawei.com.

ZHUOXIAO MENG is a Ph.D. student at the Technical University of Munich and Huawei Munich Research Center. His research interests are controllability and interoperability of large-scale simulations. His email address is zhuoxiao.meng@huawei.com.

ANIBAL SIGUENZA-TORRES is a Ph.D. student at the Technical University of Munich and Huawei Munich Research Center. His work is centered on parallel large-scale microscopic traffic simulations. His email address is anibal.siguenza1@gmail.com.

STEFANO BORTOLI is a Principal Research Engineer and Team Manager at Huawei Munich Research Center. He received a PhD in Computer Science at the University of Trento (Italy) in 2013. His research interests are traffic optimization, traffic simulations, and in general high-performance simulations. His email address is stefano.bortoli@huawei.com.

ALOIS KNOLL is a professor of Computer Science at the Technical University of Munich (TUM). He received his diploma (M.Sc.) degree in Electrical/Communications Engineering from the University of Stuttgart and his Ph.D. degree in Computer Science from the Technical University of Berlin. His email address is knoll@mytum.de.

ALESSANDRO PELLEGRINI is an assistant professor at the University of Rome Tor Vergata in the School of Engineering. He received his Ph.D. in Computer Engineering from Sapienza, University of Rome, in 2014. His research interests are high-performance computing and large-scale simulations. His email address is a.pellegrini@ing.uniroma2.it.