Online Data Extraction for Large-Scale Agent-Based Simulations

Daniel Zehe, Vaisagh Viswanathan
TUM CREATE
1 CREATE Way
138602 Singapore
+6566014015
{daniel.zehe,vaisagh.viswanathan}@tum-create.edu.sg

Wentong Cai
Nanyang Technological University
639798 Singapore
+657904600
aswtcai@ntu.edu.sg

Alois Knoll
Technische Universität München (TUM)
85748 Munich Germany
+498928918104
knoll@in.tum.de

ABSTRACT

Cloud-based simulation systems reduce the upfront hardware costs of running high-performance experiments and increases the ease with which simulation experiments can be repeated. The data being generated by simulations can be large. Commonly used data storage systems such as relational databases can handle large amounts of data, but the analysis is a challenging problem. Moreover, handling this amount of data in cloud services can be both expensive (bandwidth and storage costs) and time-consuming. However, a lot of the data that is generated by agent-based simulations does not contribute directly to the purpose of the experiment being conducted. We propose an extension to cloud-based simulation systems that rather than storing raw simulation output data, uses stream data processing to generate the result dataset while the simulation is running. This can then be used to store only the data required for later use, this saving both time and money.

CCS Concepts

•**Computing methodologies** → **Modeling and simulation; Data assimilation;** *Agent / discrete models; Simulation environments; Simulation tools;*

Keywords

Online Data Extraction, Time-variant Relational Algebra, Cloud-based Simulation, Agent-based Simulation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGSIM-PADS '16, May 15 - 18, 2016, Banff, AB, Canada

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-3742-7/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2901378.2901384

1. INTRODUCTION

The computational resources required for modern urban system simulations [18, 4] are increasing; this has also resulted in more high resolution output data being generated. Moreover, availability of much cheaper storage options[1] has resulted in data being generated at unprecedented velocities, volumes and varieties [19, 8]. Depending on the run time of a single simulation, the design decision is often taken to record as much as possible in order to reduce the number of simulation runs.

However, despite the price reduction in persistent storage, analyzing large amounts of data can still be problematic. Large data sets have to be loaded into the memory of one or multiple machines in order to do the necessary post-processing of the data. If the main memory of the system is exhausted, a distributed or stream-processing workflow has to be used. This adds an additional overhead for the user. There is a need for more efficient methods for handling the large datasets that are generated, especially with the increasing availability of cloud computing resources and cloud-based simulation services [21]. As stated in [13], "transferring data-sets to a centralized machine is thus expensive (due, for example, to network communication and other I/O related costs)".

A possible solution to this problem is to do the data analysis while the simulation is running and store only the processed data set necessary for the given experiment, i.e. the *result data set*. Such a solution would be most useful for large workflows where, traditionally, huge amounts of data have to be transfered between two consecutive steps. A possible approach to this would be to leverage on the *IEEE 1516* High Level Architecture (HLA) [6] which is popular in the simulation community. A data processing federate could be created which collects the data from the running simulations, and processes and writes it to persistent storage as required. However, the limited data transfer rate and the large overhead of publishing all information either reduces the simulation performance significantly or necessitates a larger simulation "cool-down" phase during which all data is analyzed and transmitted to a single data analysis federate.

[1] http://www.mkomo.com/cost-per-gigabyte-update

Most simulation experiment post-processing workflows start with obtaining data sets from databases, but since the size of these data sets is increasing steadily an online data extraction methodology as presented in this paper can be useful. We describe formally the components and advantages of an online data extraction methodology for agent-based simulation (specifically cloud-based simulations), in which data processing at simulation time is used to minimized the use of slower persistent storage. Using a relational algebra context of modelling the data output of a simulation does not break the post-processing workflow of many experiments. Agent-based simulations are have been taken as an example, since a set of agents of the same (or similar) type can be interpreted as a table or relation.

Using a relational algebra to model the data output of a simulation is beneficial since it does not interfere with the established post-processing workflot of many experiments. Agent-based simulations are a good example to use a relational representation of the output data, since a agents can be grouped together and be seen as a relation with tuples of state-variables.

As part of the literature review, we explore the current research in big data and stream processing as well as data description languages. Subsequently, we introduce the structure and overall working of a data-analysis component for a cloud-based simulation system. This is followed by a formal relation-algebra-based description of the data output, but unlike traditionally relational algebra, can be used to describe time-variant data. Finally, we use two traffic simulation examples to demonstrate the working and advantages of the proposed system.

2. RELATED WORK

2.1 Relational Data Description

Relational data models are widely used in the database domain where they are used to describe data by using tables from which data can be accessed and operations be executed on. The data entries are called *tuples* and share the same structure (i.e. fields) within a single database table. The advantages of the relational data description language and relational algebra are that they enable the analysis and optimization of complex and large amounts of data by using formal mathematical methods. Despite the limited number of operations, complex manipulations can be created on structured data. Relational databases were first introduced by Codd [5] and later refined by Darwen and Date [7]. In this paper, we leverage on the well-developed work in relational algebra, with a temporal component extension in order to deal with time-variant data sources that characterize simulations.

2.2 Big-Data and Stream Processing

Ranja [13] commented on the fact that the data that is being generated on the internet has been and will keep increasing rapidly over the next couple of years. This is similar to the data being generated by simulations. Individual simulations produce a lot more data than the average node on the internet. Cloud-based simulation services [22] can result in multiple concurrent simulations in the cloud at unprecedented rates which will increase the data output significantly. Ranja [13] postulates that relational databases will be unable to cope with the massive data anymore. In contrast, state-of-the-art data mining algorithms work on main memory. However, main memory is much more expensive and invariably inadequate to store the amount of data generated.

Another point to be considered is the network and I/O costs involved in the transfer of data between the storage location and the analytics computer [13]. Ranja proposes an ecosystem for data processing that involves a high velocity *data ingestion layer* that communicates with the actual *data analytics layer*, which then pushes the resulting data sets to a *data storage layer*. This kind of data analytics frameworks have different advantages and drawbacks. While the Apache Hadoop [17] distributed stream processing toolchain is more suited for historic/existing data analysis, Spark [20] and Storm streaming processing are better suited for online data-streams with highly variable (in terms of amount and type) data.

For general big data processing, different tools already exist. Tools like Apache Mahout[2] and GraphLab[3] have a large number of implemented data analysis algorithms available for use. These offer an easy-to-use off-the-shelf experience for researchers. On the other hand, building a system with such tools for simulation data analysis, that might be outside the realm of standard big data processing, presents its own set of challenges. A distributed system that relies on message passing and queuing for a general purpose application programming interface (API) is more applicable for simulations. Apache Kafka[4] for distribution of incoming data streams is an example of such a system. This usually works in conjunction with Hadoop, Storm[5] or Spark[6] to distributed high-velocity data processing payloads. Such data needs to be stored in a NoSQL database structure like MongoDB [3] or Casandra [11]; in these systems large amounts of data are stored in easily accessible structures.

Babcock et al. [2] give an overview of data streaming models and current issues in data stream processing systems. They acknowledge that there are different groups of data streaming models that handle data differently and differs from conventionally stored data in (1) availability (online vs. offline), (2) order of data elements, (3) undefined or unbound size of the stream and (4) the unavailability of historical data, since after processing the input data is discarded.

2.3 Data Extraction Techniques for Simulations

For many simulations the data export focuses on writing to a comma-separated, XML-based or application-specific binary data format. This might be acceptable for small amounts of data coming from small-scale simulations or rarely updated variables, where the final amount of raw data is rather small. For large-scale simulations, saving raw data might be desirable but infeasible due to I/O constraints. Many simulation tools, output only aggregated values or rely on visualization to transport information in the form of recordings (animation or video) or images. Prominent examples are material simulations, where the forces on a digital workpiece is shown as an overlaid heat map [10, 16].

[2]http://mahout.apache.org
[3]http://graphlab.org
[4]http://kafka.apache.org
[5]http://storm.apache.org/
[6]http://spark.apache.org

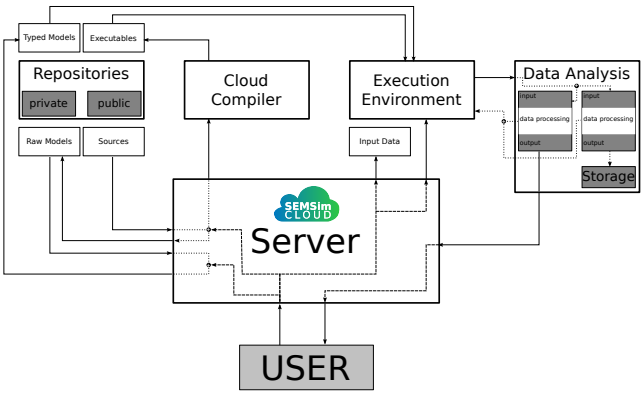

Figure 1: Cloud-based Simulation Reference Architecture. The user interacts with the system through RESTful APIs and the entire system runs on public or private cloud instances.

Schützel et al. [14] describe a stream-based reference architecture for a data management system that interacts with all the components of a simulation workflow. The described approach starts from the experiment setup, the actual execution of the simulation up to adding a "processing graph" and a storage engine. Data management has a mediating role between all steps of a simulation and passes data between the different steps of an experiment. This mediator role is also active for different simulations within one experiment run.

In a different publication, Schützel et al. [15] have presented that the extraction of simulation data is dependent on the structural dimension of the simulation entities as well as the sequential dimension that describes the order in which the data is being generated. They describe the *ML-Rules Data Extraction Language* and also the *SystemXtract Language*. The latter one uses sequential logs to reconstruct the structural and sequential information of the simulation state and passes it to the analysis application.

3. ONLINE DATA PROCESSING SYSTEM

We have previously presented SEMSim Cloud Service, a simulation cloud service for agent-based simulations with a special focus on traffic simulations [22]. In this paper we use the SEMSim Cloud Service as the reference model for giving context and for describing the proposed model. However, the methods developed are not SEMSim specific in any way. In the next section, we give a brief overview of the Cloud Service [22].

3.1 SEMSim Cloud Service

The cloud service consists of 6 main components as shown in Figure 1, of which the *Data Analysis* component is the focus of this paper.

1. The **user-interface** is represented through *representational state transfer* (REST) APIs. Through the use of APIs and non-fixed front-end, the user can tailor a use-case specific front-end for the simulation experiment.

2. These API calls are then translated by the **cloud service server** component which executes cloud service actions like allocation and deallocation of cloud resources as well as starting, monitoring and stopping of simulation experiments.

3. A **repository** for models (algorithms with parameters), types (parameter configurations for models), source code (for implemented models and simulation engine) and executables (compiled source code) is the main component of the cloud service. It can offer private repositories to privileged users or publicly available repositories to all user. This allows different domain experts to develop, validate and test models before being used by a non-domain expert.

4. The **cloud compiler** takes the source code from the repository and compiles it into an executable that can be used by the execution environment.

5. An **execution environment** is the heart of the cloud-based simulation service. It executes an executable from the repository together with parameterized models and input data to form a simulation run.

6. **Data Analysis** is an integral part of any simulation experiment and is connected to the execution environment to use the data being generated by a simulation run to deduce results by analyzing it directly. Data storage or forwarding the data stream to a visualization can range from megabytes to terabytes for a single simulation run, depending on the simulation experiment.

The complete system relies on public cloud resources allocated from cloud service providers (e.g., Google Compute Engine[7], Amazon AWS[8], Microsoft Azure[9]). These resources, mostly virtual machines but also virtual networks and virtual storage, are allocated according to the specifications of the experiment and deallocated after the experiment has finished. The data generated is stored in the cloud as well. In order to guarantee the security of the input data into the simulation, all data is encrypted until they are used in the simulation itself. Generated output data can also be encrypted, if required. The focus in this paper is on the data analysis part which is described in more detail next.

3.2 System Design

In the proposed system, the simulation instance's output data is first formally defined in a relational algebra as presented in Section 4.1. This enables us to leverage on the effectiveness and power of relational algebra in our simulation data manipulation. Once the simulation is running, a high performance simulation can output data in several ways. The naive and standard approach is to output all state variables of all agents whenever they change. However, by doing so, the bandwidth required would be very high (as illustrated in Section 5.1). A better strategy would be that external programs can subscribe to updates of variables as required for the analysis. This means that subscribers either get an update of the variable whenever it changes, at certain intervals or, if sufficient for the analysis, just once to initialize. This is a contrast to receiving always all data,

[7]https://cloud.google.com/compute/
[8]https://aws.amazon.com
[9]https://azure.microsoft.com/

whenever a state changes in the simulation. A constraint is that the update interval, in which state changes can be received, should not be smaller than the time between two events that change an attribute in the simulation (e.g., if the simulation event occurs every 1s then an update frequency of 500ms is not allowed).

This data-stream is then received by stream processing scripts. As the data is received in batches (e.g., every 1 second), and in temporal order[10], an event-based processing framework, that instantiates a new thread/process is very favorable. This also allows for scaling the processing to multiple nodes, due to its limited dependency.

Once the data has been processed by the scripts (which may even be a chain of different processing steps), this final result data set is either stored in persistent storage, or sent to a visualization client.

In order to extend a simulation engine in the cloud with a stream processing engine, the simulation engine needs to provide an API that exposes the state of the simulation to an external data sink. This API should present the data in such a way, that once requested, each change in the agent's state will be transmitted to the requester. This process continues for as long as the simulation is running, or until the request is specifically terminated by the requester. Additionally, the data sink should have some form of caching to reduce the amount of data being transfered at every change. The actual processing can be done using existing stream processing tools like Hadoop or Spark, as introduced in Section 2, which either distributes the data to be processed to one or multiple worker nodes.

3.3 Implementation

The data request scripts, that define what data is streamed from the simulation is, for ease of use, in an SQL-like language called *SimuSQL*, to ensure compatible with offline analysis of the data. An exemplary query on the output data-model of a traffic simulation requests data of electric vehicles (EV) that have the time variant property *State-of-Charge* (SOC) at an update rate of 1 second.

```
SELECT * from EV WHERE SOC>0.3 AT 1 SECOND
```

The extension `AT 1 SECOND` is added to specify how often an update of the data should be provided by the simulation. When omitting this parameter, a new data item is sent whenever available. The related concepts are described in more detail in Section 4. The data analysis middleware should take care of static/non-changing content and inject this accordingly, to ensure that the simulation can omit that data if unchanged – transparent caching.

The most important part of the cloud service, in the context of the proposed data analysis system, is an interface which the data analysis middleware provides, that the processing framework can access and request data from . This interface needs to be separate from the simulation itself, but should use locality in the data center to ensure fast data transfer. The middleware accepts requests in the above defined *SimuSQL* and returns data when available. It also takes care of the transparent caching of the data. The connection to the simulation has to be via open socket connections that support fast data transfers.

[10]note that the transport protocol can change the receive order

As can be seen in Figure 2 the data-processing starts with calling the `SELECT` statement towards the data analysis middleware component (step 1). Note, that the user initiating this data-processing work flow can be different from the user that designed or started the simulation experiment. The select statement is the trigger to instantiate a connection between the simulation and the data analysis middleware. For this specific call, the first callback will be towards the initialization function of the data analysis middleware (step 2). This is different from any other succeeding callback, because the simulation will transmit the entire current state. This includes, unlike to all other callbacks, the static data as well. All following callbacks will only include the time-variant data, as well as the key to identify the tuple unambiguously (step 3). Should a new agent be added to the simulation and should it satisfy the conditions stated in the initial request, it's data will be fully included. The data analysis middleware has to ensure that the static data from the initial data set is merged with the time-variant data stream. After the data has been prepared, the entire data set is presented to the stream processing toolchain. At the end of processing, data can be stored as a result data set or streamed out to a different data-sink. Since there is no historical data injected into the stream of data, the processing component needs to take care of passing values from one step to the next. The data will be sent from the simulation to the data analysis middleware and passed to the processing component until a cancel request is sent (step 4).

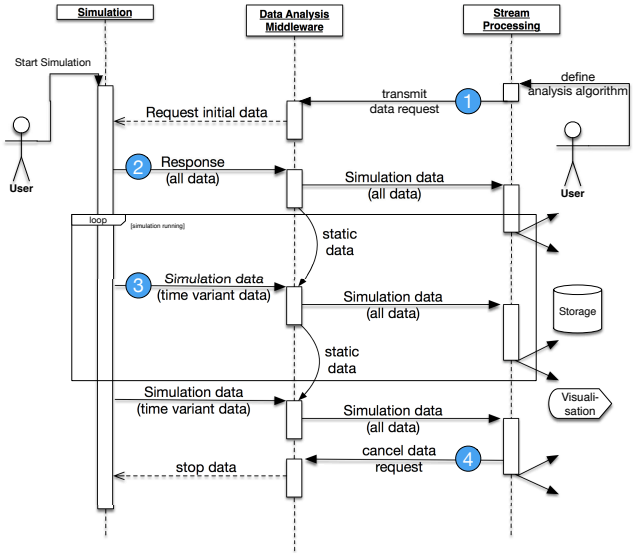

Figure 2: Sequence diagram depicting the workflow of the data analysis including the data processing. The Data Analysis middleware combines the time-variant data and the static data and passes it on to the analysis processing. Finally the result is send to storage or visualization.

The data transmitted from the simulation is not a constant stream of data. Depending on the execution speed of the simulation itself, the output will change once the state of an agent changes or a new agent is added to the simulation. For example, the position of all vehicles in a traffic simulation, will be updated rather frequently (in the range

of a few seconds), while the occupancy of a car park will be updated more infrequently (in the range of a few minutes)[11]. The data analysis middleware needs to know when all data is in the same logical time step in order to pass this block of data to the processing component. In order to handle this, *landmarks*, a concept proposed by Gama et al. [9], can be introduced into the stream by the simulation. These are meta information on which the receiver, in our case the data analysis middleware, can deduce the end of a data concurrent stream of information.

Figure 3 shows how the landmarks are inserted into the data stream of serialized data objects from the simulation. Each rectangle represents information from a specific agent. The *Entity Types* are different classes in the object-oriented framework in a simulation. The order in which entities of different types are put on the stream can vary (see A and B in Figure 3), as long as there are no 2 entity updates of the same type between 2 landmarks. It can also happen that for different entity types the amount of data varies. Also, not all entities need to be updated between two landmarks (compare B and C in Figure 3). The data-analysis middleware needs to keep track of this. It is done to ensure the reduction of bandwidth usage, by avoiding unnecessary transfers of information.

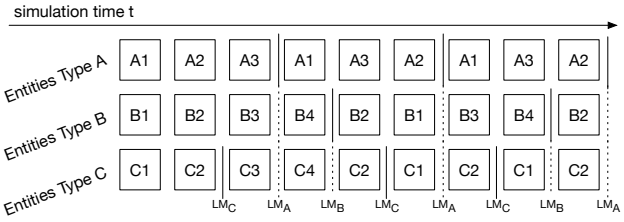

Figure 3: The Simulation inserts landmarks into the stream of updates of state changes for each entity type. The data analysis middleware detects the landmarks and discards all old information from the changed entities and replaces it with the newly received. Should an agent's state not change, the previous value is kept by the data analysis middleware.

To ensure data consistency between the simulation data model and the output data model, the landmarks play an important role. The entities that have not been updated since the last landmark, their old values are still to be considered valid. Between different entity types the time stamp of the landmarks for each type is compared by the middleware component and a consistent set of data is presented to the processing component.

4. FORMAL SIMULATION DATA REPRESENTATION

The data in an agent-based simulation can be described in multiple ways. In this paper we propose that this data be in relational algebra. This provides the advantage that constructs from a *structured query language* (SQL) can be used to retrieve data from the simulation or to store the data in a data sink (for a relational database with the same structure). In agent-based simulation, each set of agents of the same type can be perceived as tuples in a relation of a database. Therefore, the approach representing the output data-model in the form of relations is favorable, since many (post-processing) algorithms and work-flows already work on database structured data as input. Should there be a very heterogeneous agent-population, where agents can not be grouped together into relations, each agent will then form its own relation with only one time-variant tuple.

Since some of the data will change over the course of the simulation, a temporal component needs to be added to the relational schema. Mahmood et al. [12] have given an overview of how to encode temporal information into a relational data model with a focus on database use-cases. This approach is not restricted to databases and can be applied to generic relational data. This temporal relational data model can be adapted to be used in describing output data from an agent-based simulation. The difference here is that there is no (or only limited) data of the past available. Future states can be seen as starting or ending points of data output. The data analysis middleware will only receive data from the simulation when the requested time is reached. This means, in practical use, a request on a future state of an agent in the system can trigger an event to start outputting the data at that event and not sooner.

4.1 General Simulation Data Representation

The general definition of relational data models, is to characterize the well-defined structure of databases, called a database schema. One could describe the state variables of the agents in an agent-based simulation as a schema with time-varying data. The formal description of an n-ary tuple of states is usually done by writing the name of the tuple (class name) followed by a comma separated list of the state variable names in parentheses [5]. Primary keys describe a field or a set of fields which characterizes the tuple unambiguously. This primary key is indicated by underlining the respective variable names. For the temporal dependency as described in [12], an extra field is added to the relation to describe its temporal activation (when the value is valid). It is also possible to add a field for each variable of the relation, but this increases the number of variables in a relation significantly, since for each variable field an extra field for activation is required.

Since we do not have to worry about the actual time when a state variable is active, we only need to indicate that a field might change its value over the course of the simulation adding an extra field for each of the state-variable is unnecessary. We are proposing the indication of time-variance by adding that the variable is a function of time (e.g. $variablename(t)$).

$$A(\underline{f_1, f_2, ..., f_k}, f_{k+1}, f_{k+2}, ..., f_{k+n}, f_{k+n+1}(t), ..., f_{k+n+m}(t))$$

shows agent A with $n \in \mathbb{N}_{>0}$ static state variable names, $m \in \mathbb{N}_{>0}$ time-variant state variable names and $k \in \mathbb{N}_{>0}$ state variable names that unambiguously describe one $(m + n + k)$-ary tuple.

Time-variant Relational Algebra

Since traditional relational algebra has no concept of time-variant fields or tuples, the individual simple functions like projection (Π), selection (σ), Cartesian product ($\times$) and nat-

[11]Note that all time information is simulation time, not real-time

ural join ($\bowtie$) have to be translated to equivalent time-variant operations. In all following operations the n-ary tuple for a given relation has to be determined, such as that the tuples that are included in the result set before doing the algebraic function are the most recent tuples for a given primary key in respect to the time τ given. This is also expressed in Equation 1.

$$\begin{aligned} TS(id_i) &: \text{all Timestamps for a given } id_i \\ ID(ts_i) &: \text{all IDs for a given } ts_i \\ \alpha &: \Pi_{a_1,...,a_n}, \sigma_{a\theta v}, \bowtie, \times, ... \\ Dom(\alpha(R(\tau))) &: \text{Domain of operation from } \alpha \\ &\quad \text{on relation R at time } \tau \\ Dom(\alpha(R(\tau))) = &\big[\forall A(id_1, t_1) : [t_1 \in TS(id_1), t_1 \leq \tau \\ &\quad \wedge \nexists t_2 \in TS(id_1), t_1 < t_2 \leq \tau] \\ &\quad \wedge [\nexists A(id_2, t_2) : t_2 \in TS(id_2), t_1 < t_2 \leq \tau \\ &\quad \wedge id_1 = id_2]\big] \end{aligned} \quad (1)$$

Table 1 shows two exemplary relations that are used to demonstrate the time-variant operations in this section. The column TS is the time-stamp of the respective values.

Table 1: Example Relations

Relation1				*Relation2*		
ID	**V1**	**V2**	**TS**	**ID**	**V3**	**TS**
1	A	C	0	1	E	0
2	B	H	0	2	F	0
1	J	C	1	1	G	1
3	C	D	2	2	L	1
2	I	C	2			

Time-variant Projection

A projection $\Pi_{a_1,...,a_n}(R)$ returns a set of data items that contain the components $a_1, ..., a_n$ of relation R and disregards all other fields in the set of tuples. The time-variant projection includes indication for what time τ this operation should be applied. $\Pi_{a_1,....,a_n}(R(\tau))$ returns all n-ary tuples from the time-variant set R that satisfy the result of Equation 1. The projection in Table 2 outputs the variable V1 and the ID at time $\tau = 1$ from Relation1.

Table 2: $\Pi_{\mathbf{ID,V1}}(Relation1(1))$

ID	**V1**
1	J
2	B

Time-variant Selection

A selection $\sigma_{a\theta v}(R)$ returns a set of data items from the set of n-ary tuples R that satisfies the restriction expressed by attribute a, the binary operation $\theta \in \{<, \leq, =, \neq, \geq, >\}$ and the constant value v. The time-variant version of a selection needs to include the indication for what time τ this operation should be applied. $\sigma_{a\theta v}(R(\tau))$ returns all tuples that satisfy the restriction, but are also part of the result set generated by Equation 1. The selection in Table 3 outputs the variables of Relation1 with the condition that variable V2 is equal to D.

Table 3: $\sigma_{V2=D}(Relation1(2))$

ID	**V1**	**V2**
3	C	D

Time-variant Natural join

A natural join $R \bowtie S$ returns a set of data item from the sets of tuples R and S that have at least one matching attribute $\exists a_r \in R \wedge \exists a_s \in S; a_r = a_s$ in a resulting tuple. The time-variant version of this natural join includes indications for what times τ_S and τ_R this join should be applied. $R(\tau_1) \bowtie S(\tau_2)$ return all tuples as the standard join would, but under the constraint that the relations R and S are first reduced to the result generated by Equation 1. The natural join in Table 4 joins Relation1 and Relation2 at $\tau_1 = 1$ and $\tau_2 = 2$.

Table 4: $Relation1(1) \bowtie Relation2(2)$

ID	**V1**	**V2**	**V3**
1	J	C	G
2	B	H	L

Time-variant Cartesian product

The Cartesian product of $R \times S$ returns a set of data items from tuples R and S in which there is no matching attribute name $\nexists a_r \in R \wedge \nexists a_s \in S; a_r = a_s$. The time-variant version of the Cartesian product needs to include in what times τ_1 and τ_2 the tuples need to be included in the operation. $R(\tau_1) \times S(\tau_2)$ returns all standard Cartesian product tuples that are part of the result set that satisfy the Equation 1 for τ_1 and τ_2. The Cartesian product in Table 5 joins Relations1 and Relation2 at $\tau_1 = 1$ and $\tau_2 = 0$.

Table 5: $Relation1(1) \times Relation2(0)$

ID	**V1**	**V2**	**V3**
1	J	C	F
2	B	H	F
1	J	C	E
2	B	H	E

This extension to the standard relational algebra can be used when the data in a database contains additional information about the time of activation of a respective tuple.

5. CASE STUDY

As case studies for strengthening the need to reduce the amount of data stored in persistent storage, we are first evaluating the speed in which data needs to written to a storage medium and what current technologies support how many agents and simulation speeds. Secondly, we are giving an example on how the above described workflow can be used for an actual cloud based traffic simulation experiment.

5.1 Data Amount Model

In the following section we compare the amount of data that is being generated by a simulation and try to fit it to a suitable data processing framework. For this we consider the writing speed of conventional storage mediums like HDD

or SSD as well as network-based distributed data processing solutions. First, we develop a model to estimate the amount of data that is being generated by a simulation.

For this we assume an agent-based simulation with $n_A \in \mathbb{N}_{>0}$ number of agents, the update frequency of the agents t_s (in seconds), the run time of the simulation T (in seconds) as well as the function $g(\tau)$, the percentage of agents being updated at given time. Additionally, the run time performance as a *real time factor* RTF (how much faster than real-time) needs to be considered as well as the number of state variables $s \in \mathbb{N}_{>0}$ each agent can have. An exemplary function $g(\tau)$ can be seen in Figure 4.

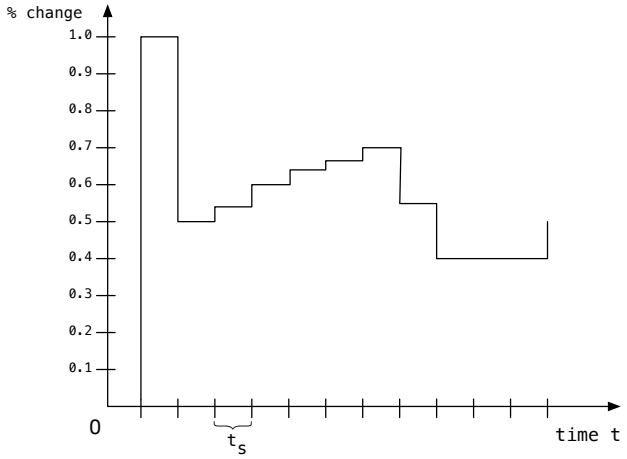

Figure 4: Time-variant percentage of agents changing in the simulation. t_s describes the time between subsequent simulation steps.

The final mathematical model to determine how many values will have to be saved or transmitted is:

$$TotalValues = \sum_{t=t_0}^{T} g(t) * n_A * s \tag{2}$$

All variables are double values on a 64 bit system, the memory footprint is different from the space required when outputting into CSV format. The precision $p \in \mathbb{N}_{>0}$ needs to be regarded as well, since this will influence the number of bytes that need to be written to disk. If the precision is 10 digits, a string representation of the number with ASCII coding would need 11 bytes, due to the decimal point. This is 3 bytes more than in the computer's main memory.

$$CSVsize = TotalValues * (p + 1) \tag{3}$$

In order to evaluate if a certain storage medium can be used, the specification of common hardware write speeds need to be collected. For this we are considering conventional hard drives with a writing speed of 80 - 160 MB/s as well as solid state disk with a writing speed of 400 - 1000 MB/s. Additionally, state-of-the-art network transfer speeds for local networks need to be considered, since the data analysis is not be done on the same machine as the simulation is being executed. Current data center interconnections technologies are Ethernet, Infiniband or fiber optical connections. They range from $10 - 40GB/s$, $56GB/s$ to $> 100GB/s$ respectively. The output bandwidth in bytes per second is defined as

Table 6: Agent count vs. writing speed

	5000	8000	80000	800000
	103.76 MB/s	166.02MB/s	1.66GB/s	16.6GB/s
HDD	✓	-	-	-
SSD	✓	✓	-	-
10G Ethernet	✓	✓	✓	-
56G Infiniband	✓	✓	✓	✓
100G Fibre	✓	✓	✓	✓

$$OutputSpeed = \frac{CSVSize}{\frac{T}{t_s}} * RTF \tag{4}$$

Conventional way of processing simulation output data also requires to read data into main memory. There are also limits on main memory of a single system. A distributed processing workflow has to be used in such cases. This allows to distribute workload to multiple worker nodes and use a single aggregation node to combine the results. this works very well, when the data has little or no dependencies. The developed model can help to design the right data analysis workflow for the agent-based simulation that is to be run.

Data amount example

In a simple example where we vary the number of agents in a 24-hour simulation with each agent having 25 state variables that change throughout the simulation. The time step is 1 second and the RTF is 100. Over the course of the simulation with each time step 50% of the agents change and their state has to be outputted. Assuming a precision of 16 digits.

This would result in $87.55GB$ of total data and 1.04 MB per simulated second of simulation time when considering 5000 agents. Multiplying it with the RTF of 100, it results in $103.76MB/s$ that have to be written to storage. This would be possible with all the listed storage options in Table 6. The introduction of more agents into a simulation will have the effect that the simulation data can not be written out anymore and the systems capability is exhausted.

In the presented system, the large data would only be transfered from the simulation to a distributed processing workflow.

This mean the large amount of data generated by a simulation is streamed to the processing systems using high throughput network communications. Since the processing system might be distributed, even a lower capability virtual machine can handle the incoming connections. This allows for shorter turnaround times, between the end of a simulation, since a hard-disk-based approach would need to wait until all data is written before reading it from the storage medium. Once the simulation output data bandwidth exceed the capability of the medium, the I/O presents a bottleneck to the simulation experiment.

5.2 Traffic Simulation Cloud Service

The SEMSim simulation engine [1, 18] is used in the SEMSim traffic simulation cloud service. The *Scalable Electromobility Simulator* (SEMSim Traffic) is an agent-based traffic simulation engine. It is part of the SEMSim platform, which also includes a power system simulation [4]. This allows researchers to study the holistic effects of electromobility on an entire city/region through simulation.

Within the SEMSim traffic simulation each agent is represented as driver-vehicle-units (DVU) which consists of driver behavior and vehicle component models. Depending on the specific agent the actual models can vary.

The simulation uses a hybrid time-stepped and event-based execution model. Specific events like agent movement have predefined intervals, whereas other events, like the decision making of the driver or the update of the air-conditioning, are scheduled at different intervals. This gives the flexibility to vary the update of certain models more frequently than others.

An exemplary simulation experiment [4] with electric vehicle agents in a city-scale network and the objective is to determine the locations and State-of-Charge (SoC) of all electric vehicles that have an SoC of less than 0.3.

5.2.1 Traffic Simulation Data

In a nanoscopic agent-based traffic simulations like SEMSim Traffic, the agents are the vehicles and their driver. The behavior of the driver directly influences the state of the vehicle. For example, the decision by the driver model to increase the velocity directly influences the acceleration, fuel consumption and, ultimately, the velocity of the vehicle itself. Since the vehicle is not the only entity in a traffic simulation, the output data representation also needs to consider other entities:

- *Roads*: Consist of links and lanes [18], used for routing and movement.
- *Car Parks*: Hold a certain number of vehicles.
- *Traffic Lights*: Regulate the traffic at intersections and control the flow of vehicles.
- *Other Infrastructure Components*: Bus Stops, Crosswalks, Tram/MRT stations

An agent-based traffic simulation is usually modeled in an object-oriented framework, all the entities can have inheritance to child-models with larger amounts of data (see Figure 5). A vehicle only contains the very basic attributes like speed, location and geometric dimensions, while more specific vehicle implementations (e.g., electric, fuel-cell, ICE) can have more specific model attributes that influence the speed or location. The same applies for driver behavior. The output data relations for this example, containing only a subset of all entities in the simulation, would be:

```
Vehicle(vehicleID,geometry,
    velocity(t),location(t))
EV(vehicleID,batterySize,currentCapacity(t))
ICE(vehicleID,MotorPower,tankSize,
    tankfilled(t))
Road(ID,name,startpoint,endpoint)
Link(ID,startpoint,endpoint)
Lane(ID,startpoint,endpoint,leftlane,rightlane)
CarPark(ID,location,totalSlots, freeSlots(t))
Trip(ID,vehicleID,FromLocation,ToLocation)
```

This relation scheme needs to be known to the user that is developing the processing scripts, because these are the relations a *SimuSQL* query can return results for.

5.2.2 Time-variant Relational Data Model

The data required from the simulation is the vehicle location as well as the information of the SoC. A time-variant relation algebra expression would look like the following when

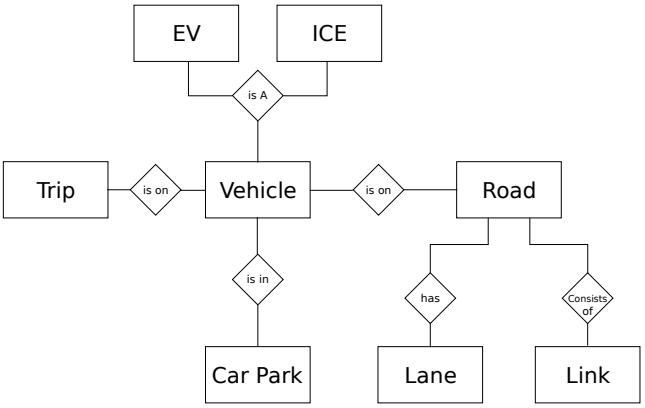

Figure 5: ER output data Model for a agent-based traffic simulation

considering that t is the current time t_{now}:

$$\text{LowSOCAgents}(t_{now}) = \text{Vehicle}(t_{now}) \bowtie \sigma_{currentCapacity/batterySize<0.3}(EV(t_{now})) \quad (5)$$

$$\Pi_{vehicleID,location,batterySize,currentCapacity}(\text{LowSoCAgents}(t_{now})) \quad (6)$$

This projection will return the vehicle ID, the location of the vehicle, the current capacity of the battery as well as the total battery size. Due to the selection in equation 5, the natural join only includes those agents that have an SoC of less than 0.3. The SoC is defined as the current capacity of the battery over the maximal battery size. The equation 6 applies a projection on the result tuples of equation 5. This is the formal description and operations that can be used to optimize queries using time-variant relational algebra. The following section expresses the same query as an *SimuSQL* statement which is queried from the processing script to the data analysis middleware.

5.2.3 Stream Output for Traffic Simulations

After we have introduced the data that a traffic simulation is generating, and the formal time-variant relation algebra optimization that has been performed, the translation into a *SimuSQL* query and the workflow at the data analysis middleware will be shown now.

A *SimuSQL* query to obtain the location and the SoC of all vehicles in the simulation that have less than 0.3 would be:

```
SELECT vehicleID,Location,currentCapacity/
    batterySize as SOC from EV natural
    join Vehicle WHERE currentCapacity/
    batterySize < 0.3;
```

This will return a time-variant tuple of all vehicles containing the location as well as batterySize as a fixed value whenever a vehicle has an SoC (currentCapacity/BatterySize) of less than 0.3. Since there is no update interval (e.g. `AT 1 SECOND`) specified, the data is streamed whenever changed in the simulation.

This query will be forwarded to the data analysis middleware where it is translated into a statement requesting information from the vehicle object as well as the inherited properties from the EV object in the simulation. The

first request will transmit the entire state of the simulation regarding the vehicles and the EVs to the data analysis middleware. The transparent caching structure in the data analysis middleware will then cache the time-invariant variables, like the `batterySize` in this case and pass the entire data stream to the processing engine. Here the processing of spatial information of vehicles is performed and the result of clustering "low SoC agents" is transmitted to the database and/or pushed to a visualization engine, which eventually displays this information. In parallel to the processing, the simulation continues running and data continues being sent to the data analysis middelware. With any subsequent stream of data after the initial one (containing the `batterySize`), only the primary key, in this case the `vehicleID`, together with the time-variant information (e.g. currentCapacity) is forwarded. In the above example, data analysis middleware takes care of inputting the `batterySize` from the transparent cache and forwards it to the processing engine. To the processing engine, the simulation looks like a normal SQL database; however, in the background the simulation is changing the data constantly, while the cloud service is trying to minimize the data transmitted between the simulation and the data analysis middleware and the processing engine.

6. CONCLUSIONS

In this paper we have presented an approach to reduce the data generated by simulation experiments and to couple the data analysis to the simulation execution. This is especially useful for cloud based simulation experiments, where the costs of running simulation experiments is low, but long term storage and transfer of large amounts of data can be really expensive. The proposed system uses stream processing of data during its generation and a relational data model of the possible output data generated by a simulation experiment. This enables experiment designers to formally model the simulation output data with a time-variant relational data model.

We propose a formalism that indicates time variance on a simple relational model, where all operations have to regard the time at which the data was created. Since this method introduces another layer of middleware into the simulation work-flow, the middle-ware component presents a possible bottleneck to the system. This needs to be evaluated in experimental studies, but since the middle-ware discards any historical data, the execution, even of complex queries, is expected to be fast.

Possible use-cases of this model is an online data visualization application that can show results, and possible interactions with the simulation while it is still running. (Semi-)Manual exploration of different simulation and model configurations could be much more engaging and fruitful. This can be used for visualization of the data but also for decision support systems that deliver results while the simulation is still running.

7. REFERENCES

[1] H. Aydt, Y. Xu, M. Lees, and A. Knoll. A multi-threaded execution model for the agent-based semsim traffic simulation. In G. Tan, G. Yeo, S. Turner, and Y. Teo, editors, *AsiaSim 2013*, volume 402 of *Communications in Computer and Information Science*, pages 1–12. Springer Berlin Heidelberg, nov 2013.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[3] K. Banker. *MongoDB in Action*. Manning Publications Co., Greenwich, CT, USA, jan 2012.

[4] D. Ciechanowicz, D. Pelzer, and A. Knoll. Simulation-based approach for investigating the impact of electric vehicles on power grids. In *Proceedings of IEEE PES Asia-Pacific Power and Energy Engineering Conference 2015*, Nov 2015.

[5] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, jun 1970.

[6] J. Dahmann and K. Morse. High level architecture for simulation: an update. In *Distributed Interactive Simulation and Real-Time Applications, 1998. Proceedings. 2nd International Workshop on*, pages 32–40, Jul 1998.

[7] H. Darwen and C. J. Date. The third manifesto. *SIGMOD Rec.*, 24(1):39–49, mar 1995.

[8] W. Fan and A. Bifet. Mining big data: current status, and forecast to the future. *ACM sIGKDD Explorations Newsletter*, 14(2):1–5, dec 2012.

[9] J. Gama and P. Rodrigues. Data stream processing. In J. Gama and M. Gaber, editors, *Learning from Data Streams*, pages 25–39. Springer Berlin Heidelberg, sep 2007.

[10] P. Kurowski. *Engineering Analysis with SolidWorks Simulation 2012*. SDC Publications, apr 2012.

[11] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, apr 2010.

[12] N. Mahmood, S. M. A. Burney, and K. Ahsan. A logical temporal relational data model. *CoRR*, abs/1002.1143, jan 2010.

[13] R. Ranjan. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, 1(1):78–83, may 2014.

[14] J. Schützel, H. Meyer, and A. M. Uhrmacher. A stream-based architecture for the management and on-line analysis of unbounded amounts of simulation data. In *Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '14, pages 83–94, New York, NY, USA, may 2014. ACM.

[15] J. Schützel and A. M. Uhrmacher. Targeted extration of simulation data. In *Distributed Simulation and Real Time Applications (DS-RT), 2015 IEEE/ACM 19th International Symposium on*, Oct 2015.

[16] S. Tickoo. *Autodesk Simulation Mechanical 2015 for Designers*. CADCIM Technologies, sep 2014.

[17] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache

hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, oct 2013. ACM.

[18] V. Viswanathan, D. Zehe, J. Ivanchev, D. Pelzer, A. Knoll, and H. Aydt. Simulation-assisted exploration of charging infrastructure requirements for electric vehicles in urban environments. *Journal of Computational Science*, 12:1–10, jan 2016.

[19] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding. Data mining with big data. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):97–107, Jan 2014.

[20] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pages 10–10. USENIX Association, jun 2012.

[21] D. Zehe, W. Cai, A. Knoll, and H. Aydt. Tutorial on a modeling and simulation cloud service. In *Proceedings of the 2015 Winter Simulation Conference*, dec 2015.

[22] D. Zehe, A. Knoll, W. Cai, and H. Aydt. Semsim cloud service: Large-scale urban systems simulation in the cloud. *Simulation Modelling Practice and Theory*, 58, Part 2:157 – 171, nov 2015. Special issue on Cloud Simulation.