SimDB in Action: Road Traffic Simulations Completely Inside Array DBMS

https://wikience.github.io/simdb2022

Ramon Antonio Rodriges Zalipynis
HSE University
Moscow, Russia
rodriges@gis.land
arodriges@hse.ru

ABSTRACT

Array DBMSs operate on big N-d arrays. Cellular automata (CA) work on a discrete lattice of cells, essentially on N-d arrays. CA facilitate decision support as they realistically simulate complex phenomena including road traffic, fire spread, and urban growth. Array DBMSs can bring numerous benefits to the CA domain via a "database approach": powerful parallelization, out-of-the box array operators, and interoperability to name a few. On the other hand, CA expand the area of Array DBMS applications and open a wide range of R&D opportunities. However, it is not straightforward to make an Array DBMS to support CA simulation workloads. SimDB enables end-to-end CA simulations directly inside the ChronosDB array DBMS via numerous new components and is the first effort to run CA simulations entirely inside an Array DBMS. We also developed a new desktop application specially designed to showcase SimDB. The application features interactive components to graphically reveal the insights of SimDB internals. Moreover, our application provides a convenient GUI to comprehensively investigate how end-to-end road traffic simulations run entirely inside an Array DBMS.

PVLDB Reference Format:
Ramon Antonio Rodriges Zalipynis. SimDB in Action: Road Traffic Simulations Completely Inside Array DBMS. PVLDB, 15(12): 3742 - 3745, 2022.
doi:10.14778/3554821.3554889

1 INTRODUCTION

Array DBMSs are rather young systems which target N-d arrays. Sophisticated storage, efficient processing, and interactive visualization of N-d arrays make a far from complete range of problems tackled by Array DBMSs [9]. This range is rapidly expanding due to advances in fundamental Array DBMS principles and the growth of volumes and diversity of N-d arrays. A contemporary survey of Array DBMSs, array-oriented systems, array techniques, and emerging R&D opportunities in the area is in [9].

Now, we complement a recent research paper that presents a novel and rather unusual application for Array DBMSs: physical world simulation [10]. In addition to naturally representing important data types from numerous domains [2], N-d arrays are also at the heart of data-driven and numerical simulations [1, 10].

Simulation capabilities are important for Array DBMSs. From a scientific point of view, simulation scenarios pose a lot of challenges and open new opportunities for making Array DBMSs more robust and applicable to a wider range of problems. From the perspective of a modeler, an Array DBMS can be thought of as a flexible framework for coordinating end-to-end simulations with a "database approach": powerful array tools that support the whole simulation lifecycle.

Practitioners heavily utilize Cellular Automata (CA) to realistically simulate road traffic, fire spread, urban growth, land cover change, and lava flow to name a few. Importantly, CA physical environments are modeled by N-d arrays. Hence, we pioneered the incorporation of simulation capabilities into an Array DBMS, namely ChronosDB [7, 8], using Traffic Cellular Automata [10].

SimDB is a collective name for all facilities required to run end-to-end CA simulations in ChronosDB [10]. Previous literature never describes end-to-end parallel CA simulations coupled with DBMS-style simulation data management using an array-oriented system [9]. In particular, SimDB provides a unique combination of numerous benefits for a CA modeler in a single system, ranging from data preparation to simulation to animation.

Although CA and Array DBMSs have a common data model, CA simulations pose sophisticated design challenges to Array DBMSs, e.g., iterations & imperative rules are inherent to CA. We developed a separate desktop application in C# with a powerful GUI to interactively uncover the insights of how SimDB solves the challenges by the first native UDF language for Array DBMSs, proactive simulation plans, novel convolution operator, and other components.

We focus on one of the most complex CA models: traffic cellular automata (TCA). The model simulates vehicle movement on multiple lanes with road intersections controlled by traffic lights. Vehicles have different lengths, moving directions, varying speed, can change lanes, directions, and overtake each other. Due to space constraints, readers can find model details in our research paper [10].

In our desktop application, we provide a graphical and engaging road network constructor, interactive maps, and statistical charts. We designed the app for high user interaction. We invite the users to undertake an end-to-end TCA simulation with SimDB. We start from initializing input simulation arrays, studying and running simulation UDFs. We then interactively explore simulation plans, debug UDFs, and compute statistics using a convenient desktop GUI. We also visualize and animate input/output arrays.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554889

2 SIMDB OVERVIEW

Now, we briefly describe SimDB, a deep modification of ChronosDB, our recent array DBMS [7, 8]. It might seem that Array DBMSs can readily run CA simulations. However, certain simulation peculiarities make CA simulations impossible in current Array DBMSs. SimDB solves key design challenges to enable end-to-end CA simulations completely inside an Array DBMS for the first time [10].

2.1 Generic Convolution Operator

Design Challenge 1. *How to support arbitrary cellular automata local transition rules in Array DBMSs?*

A CA rule looks like a well-known convolution operator [11]. However, a CA rule is much more complex: it is a procedure that (1) is applied for each cell of several input arrays, (2) checks for constraints, (3) may update several cells within the neighborhood.

To support arbitrary CA rules, we introduced a new convolution operator for Array DBMSs. Users provide the logic as UDFs in a high-level language, currently in Java. SimDB iterates over arrays, forms readonly input and writable output windows, all equipped with helper functions, e.g. rotate by 90° (adjusts the local coordinate system to use the same code for different vehicle types), fig. 1.

Unlike a traditional convolution, our operator feeds a convolution UDF several input windows and allows the UDF to modify an arbitrary number of cells within multiple output windows. The latter enables an operator to produce several output arrays.

```
public interface ConvolveWindow<T> {
   enum Degrees {_0, _90, _180, _270}
   int getArrayX(); // x & y of (0, 0) in the array
   int getArrayY();
   int xWindowSize();   int yWindowSize();
   int xSubarraySize(); int ySubarraySize();
   T rotate(Degrees degrees);
   Random random(); // random number generator
   void move(int currentX, int currentY);
}
```

Figure 1: An Interface for the Convolution Window.

2.2 Native UDF Language for Array DBMSs

Design Challenge 2. *How to efficiently (natively) support iterations directly inside an Array DBMS?*

Iterations are inherent to physical simulations. Perhaps one may code iterations in Python/C++ UDFs. However, such UDFs are black-boxes that Array DBMSs cannot optimize [5]. Existing query languages are unable to express iterations. Running iterations query by query requires a query output to be completely materialized before the next query. In addition, holistic optimizations for several iterations ahead are unavailable as the overall picture is unclear.

We introduced the first native Array DBMS language for UDFs. SimDB uses strict formal definitions of array operations [7] and compiler techniques (e.g., loop unrolling) to build and execute simulation plans for several iteration steps ahead (proactive simulation plans). This lets SimDB avoid redundant materializations and reduce scheduling overheads; e.g., "`scheduled on workers`" tasks can be executed without communicating with the coordinator, fig. 3.

SimDB UDFs are easy to code: they consist of commands with a syntax similar to command line tools: familiar to most users, fig. 2.

```
cp tca.speed $speed ↵ [newline] cp tca.length $length
val {window} "-xWindowSize 11 -yWindowSize 11"
foreach {step} -from 0 -to 100 ↵ begin
   calc tca.lane:in $speed:in $length:in tca.temperature:in \
        --overwrite $speed_move:out $length_move:out      \
        -ot Int16 {window} -classfile "TCA.java"             \
        -method_name moveForwardPhase
   ...
   calc tca.lane:in $speed_lights:in $length_lights:in \
        --overwrite $speed:out $length:out               \
        -ot Int16 {window} -classfile "TCA.java"         \
        -method_name lightsPhase
   append $speed speedh ↵ append $length lenh ↵ end
```

Figure 2: Part of the TCA simulation UDF (see section 3.3)

Design Challenge 3. *How to correctly execute a native UDF for Array DBMSs to generate a proactive simulation plan?*

Although the UDF looks small in fig. 2, it is challenging to execute. For example, during loop unrolling, the same array name appears 100 times: e.g., the last `calc` command deletes current $speed array and creates a new array with the same name. We must be able to keep and address all arrays (deleted and new) and write/read all of them simultaneously when we build and execute a simulation plan.

SimDB maintains several versions of an array with the same name during runtime. SimDB refers to an array by its name and version, operating on deleted and new arrays. A UDF acquires locks for an array name to (1) control the state of arrays (exists, deleted, newer version created, etc.), (2) prevent other UDFs from modifying arrays with the same name. Array versioning and locks are transparent to users, but required deep modifications to ChronosDB.

2.3 Simulation Scheduling

A proactive simulation plan is a DAG similar to a ChronosDB execution plan [7]. However, SimDB treats such plans differently: (1) the unit of schedule is a subgraph, not a vertice or the whole DAG to reduce scheduling overheads, (2) cluster nodes receive subgraphs and *merge* them with the plan they have so far into a single large plan to continue smooth execution. The nodes exchange array partitions; the coordinator node is not involved, fig. 3. SimDB incrementally builds simulation plans to control the memory footprint.

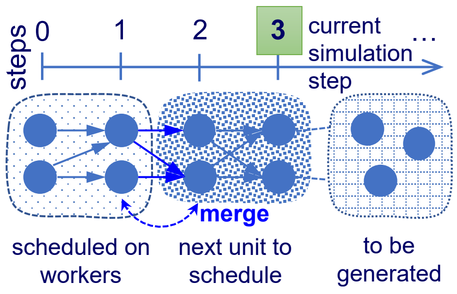


Figure 3: Proactive Simulation Plan.

3 SIMDB INTERACTIVE GUI

We guide users through a step-by-step, end-to-end CA simulation (TCA) by SimDB in its interactive GUI, fig. 4. We also showcase the benefits of using SimDB for CA simulations, e.g. interoperability, UDF debugging, and visualization. In this way, we also answer the question why SimDB is an excellent choice for this workload.

A constructor (section 3.1) lets users create a road network which is converted to input arrays and initialized with UDFs (section 3.2). A native Array DBMS UDF is used to generate simulation plans (section 3.4). It can call Java UDFs (section 3.3). SimDB can animate arrays on an interactive map and is interoperable (section 3.9).

The goal of a TCA simulation is to derive road traffic statistics (section 3.7) from history arrays (section 3.3) for decision support. SimDB facilitates simulations with its powerful capabilities.

To make user sessions with the SimDB GUI more engaging and flexible, users can start from any step (sections 3.1 to 3.9): we prepared ready-to-use examples for each simulation step beforehand.

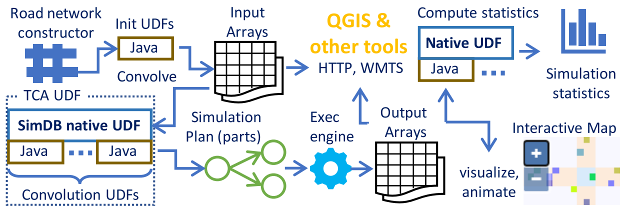


Figure 4: SimDB End-to-End Simulation Overview.

3.1 Road Network Constructor

Users start by creating the input array (or choosing an example array) that represents a road map (a 2-d array): `tca.lane`, fig. 2.

To better understand the SimDB internals, we provide a graphical road network constructor. To make it more engaging and interactive, users can draw on a real-world map. Traditionally, the road network of Manhattan, New York is used for a traffic CA model, so the map will automatically focus on this area by default.

Users can draw vertical and horizontal rectangles (polygons) that represent roads. The only constraint is that they should be orthogonal to each other (the case for non-orthogonal roads is left for future work). Once the map is ready, it is rasterized to an array. The number of lanes for the road will be proportional to the polygon width/height. Traffic lights will be placed automatically in accordance with the model definition.

3.2 Initializing the Simulation

The user continues by initializing other input arrays (using UDFs): initial vehicles' speeds (`tca.speed`) and lengths (`tca.length`) and watching results on an interactive map (section 3.6). We created a sample (editable) Java UDF to scatter vehicles randomly on the lanes, accounting for their lengths and avoiding road intersections. We model a vehicle by one cell having its rear bumper.

The initialization takes place in two phases. First, we must decide whether a cell will be occupied by a vehicle or not. Second, we must assign a length and a speed to each vehicle.

Figure 5 presents a UDF for assigning a speed. The condition checks whether we are at a cell with a West-East or a South-North moving vehicle. If the condition is true, a random speed is assigned. Otherwise, we additionally check whether we are at a cell with traffic lights to assign the remaining number of ticks.

```java
public void setSpeed(ConvolveWindows w) {
   double val = w.input(0).get(0, 0);
   Double output = null;
   if (val == 1 || val == 0) {
      output = (double) w.input(0).random().nextInt(4);
   } else {
      if (val == 4) {  // traffic lights
         int rnd = w.output(0).random().nextInt(2);
         output = (double) (200 + rnd + 1);
      }
   }
   w.output(0).set(output, 0, 0);
}
```

Figure 5: A UDF for Assigning a Speed for a Vehicle.

3.3 Exploring Native Array DBMS UDFs

The user can use sample UDFs for TCA simulation. Let us describe the main UDF, fig. 2. Note that SimDB supports command nesting.

To keep input arrays intact, we start by copying them to interim `$speed` & `$length` arrays which are subject to optimizations, e.g. they mostly reside in RAM. The loop will be executed 100 times.

The `calc` command runs the new convolution operator with a Java UDF: a `*.java` file with the specified method name. SimDB compiles the file to bytecode for faster execution. `calc` accepts/produces an arbitrary number of input/output arrays. Quantiles `:in` and `:out` distinguish between the in/out arrays as their number is not fixed. The `-overwrite` flag forces `calc` to delete, if they exist, and recreate output arrays with the same name.

An iteration ends by appending new 2-d arrays `$speed` & `$length` to 3-d history arrays `speedh` & `lenh` along the virtual *time* axis. To run a UDF, just click the respective button in the GUI.

3.4 Investigating Proactive Simulation Plans

We invite users to investigate plan visualizations interactively to better understand SimDB plan scheduling and execution. SimDB exposes its plans in the open Graph Modeling Language [3]. SimDB lays out vertices in 2-d to avoid clutter, assigns colors, and annotates them with extensive statistics: task assignment, network I/O, dataset info, etc. Just type `explain` before a UDF to get its plan. Users can zoom, pan the plan, summarize statistics, filter, highlight tasks and dependencies in Gephi, a free graph visualizer [3], fig. 6.

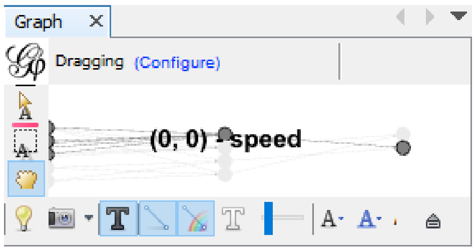
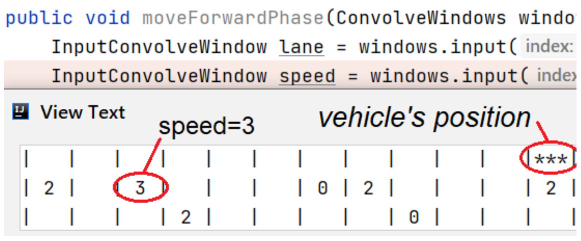


Figure 6: Exploring Simulation Plans.

3.5 Debugging TCA UDFs

To get a clear idea of the new convolution operator and TCA rules, we encourage users to debug, step-by-step, TCA UDFs in Java used for `calc` commands, e.g. `moveForwardPhase` (in IntelliJ IDEA). SimDB runs any Java files, but we put `TCA.java` in a module visible to the IDE. For a time step and a vehicle, the user can leverage the power of IDE to explore UDFs and operator windows rendering themselves as ASCII tables (nearby vehicles' speed/length/location, traffic lights) and track TCA decisions, fig. 7.



```
public void moveForwardPhase(ConvolveWindows windo
    InputConvolveWindow lane = windows.input( index:
    InputConvolveWindow speed = windows.input( index
```

```
|   |   |   |   |   |   |   |   |   |   |   |***|
| 2 |   | 3 |   |   | 0 | 2 |   |   |   | 2 |
|   |   |   | 2 |   |   |   | 0 |   |   |   |   |
```

Figure 7: Convolution Window: $speed array

3.6 Interactive Array Visualization

Visualization is essential for data understanding. SimDB provides array images via the open, popular, standard WMTS protocol [13]. SimDB WMTS supports N-d arrays, so the GUI queries SimDB to hyperslab speedh[i:i, lat, lon] (or lenh) for simulation step i and displays it on a built-in interactive map, fig. 8. A time slider makes it easy to tune i. The user can add/remove other arrays (e.g. tca.lane) to/from the map, pan, zoom & adjust their color palettes.

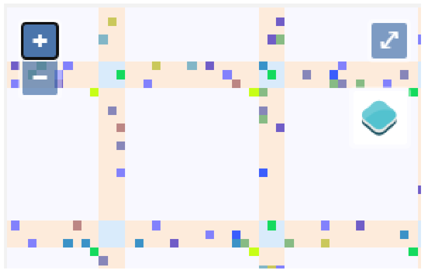


Figure 8: A Part of speedh Interactive Map.

3.7 Computing Simulation Statistics

SimDB serves all phases of the simulations within the single system: even computing statistics. We invite users to derive simulation statistics by sample queries and interactively explore results using the map. As an example, we take typical TCA statistics: Mean Traffic Density (the mean number of vehicles passed through a cell) and Space-Mean Speed (the mean vehicle speed for a cell) [4, 12].

The task is not as straightforward as it might look. This is because vehicles with length of more than 1 cell occupy only a single cell in an array. Hence, to correctly compute the statistics, arrays must be preprocessed by specialized UDFs on-the-fly.

3.8 Interactive Simulation Animation

SimDB animates history arrays, so users can watch modeling at work: how vehicles move along the roads, queue at traffic lights, turn at road intersections, change speed, and overtake each other.

Users can tune the animation speed (the time interval of switching frames) via a slider: from 0 (no animation) to 5 seconds. To start animation, just set the slider to a non-zero position. Frame f_i = speedh[i:i, lat, lon] or lenh array. Frames are displayed on the interactive map, the switch between f_i and f_{i+1} happens smoothly.

Animation plays $f_{j \bmod 100}$ for $j = 0, \infty$. When the animation is in progress, users can rewind to any frame by the time slider, turn on/off arrays, pan, zoom in/out, and go fullscreen.

3.9 Experiencing Interoperability

SimDB storage layer is built on top of raw files in standard formats. Simulation arrays are in GeoTIFF format and readily accessible to other software. **Example 1**. (1) run `ls speedh` (or lenh) to view array list for each time step. SimDB is also an HTTP-server: there is an http link next to each array, (2) use any link to download an array: a full-fledged, georeferenced GeoTIFF file, (3) open the file in a popular and free QGIS [6], zoom it, create palettes, overlay with other layers, explore cell values and metadata. **Example 2**. QGIS is also a WMTS client, so add SimDB as a WMTS layer to view an array on a QGIS map without manually downloading its partitions.

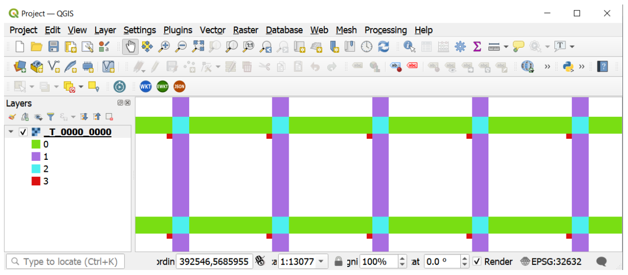


Figure 9: A SimDB Array in Quantum GIS.

REFERENCES

[1] Venkatramani Balaji, Alistair Adcroft, and Zhi Liang. 2019. Gridspec: A standard for the description of grids used in Earth System models. In *arXiv*.
[2] ArcGIS book. 2022. https://learn.arcgis.com/en/arcgis-imagery-book/
[3] GML. 2022. https://gephi.org/users/supported-graph-formats/.
[4] Sven Maerivoet and Bart De Moor. 2005. Cellular automata models of road traffic. *Physics reports* 419, 1 (2005), 1–64.
[5] Parmita Mehta et al. 2017. Comparative evaluation of big-data systems on scientific image analytics workloads. *PVLDB* 10, 11 (2017), 1226–1237.
[6] Quantum GIS. 2022. https://www.qgis.org/.
[7] Ramon Antonio Rodriges Zalipynis. 2018. ChronosDB: Distributed, File Based, Geospatial Array DBMS. *PVLDB* 11, 10 (2018), 1247–1261.
[8] Ramon Antonio Rodriges Zalipynis. 2019. ChronosDB in Action: Manage, Process, and Visualize Big Geospatial Arrays in the Cloud. In *SIGMOD*. 1985–1988.
[9] Ramon Antonio Rodriges Zalipynis. 2021. Array DBMS: Past, Present, and (Near) Future. *PVLDB* 14, 12 (2021), 3186–3189.
[10] Ramon Antonio Rodriges Zalipynis. 2021. Convergence of Array DBMS and Cellular Automata: A Road Traffic Simulation Case. In *SIGMOD*. 2399–2403.
[11] Ramon Antonio Rodriges Zalipynis et al. 2018. Array DBMS and Satellite Imagery: Towards Big Raster Data in the Cloud *(LNCS)*, Vol. 10716. 267–279.
[12] Ozan K Tonguz et al. 2009. Modeling urban traffic: a cellular automata approach. *IEEE Communications Magazine* 47, 5 (2009), 142–150.
[13] WMTS. 2022. https://www.opengeospatial.org/standards/wmts.