Cloud Based Data Analytics for Disaster Management

Matthias J. Sax — mjsax@informatik.hu-berlin.de — in METRIK since 01.04.2011

supported by DFG

Data Analysis in Disaster Management

One of the challenges of data processing in the domain of disaster management is the rapid growth of collected data and the requirement for shorter processing times. Wireless sensor networks deployed in a city (e.g., as an earthquake early warning system) may span thousands of sensor nodes resulting in gigabytes of collected data per minute. Currently, the collected data is stored to disk and analyzed offline because of long processing times. However, many application require a near real-time analysis what is not feasible with current systems.

Another example is the evaluation of flood simulation data at the **Helmholtz-Zentrum Potsdam (Deutsches GeoForschungsZentrum GFZ)**. Geoscientists run flood simulations with different parameter settings, before applying data mining techniques to the generated data to better understand the input-output-correlation. Usually, data mining techniques are very compute intensive. Right now we do not process the given flood simulation data in a reasonable time.

We propose to use parallelization techniques for data-flow programs to address the problem of long processing times caused by complex analytics and/or huge amount of data.

Parallel Execution of Data-Flow Programs

The MapReduce approach is a major contribution for designing a parallel data-flow system which scales up to a big number of computers. The Stratosphere system generalizes the MapReduce approach by introducing the PACT programming model (PACT: Parallelization Contract). Figure 1 shows the basic PACT design: A PACT operator consists of a system provided second-order function (Input Contract) and a user-defined imperative first-order function (UDF). The Input Contract determines how the UDF is applied to the data. A PACT program is an acyclic-directed graph in which the nodes are PACT operators and the directed edges define the data-flow.

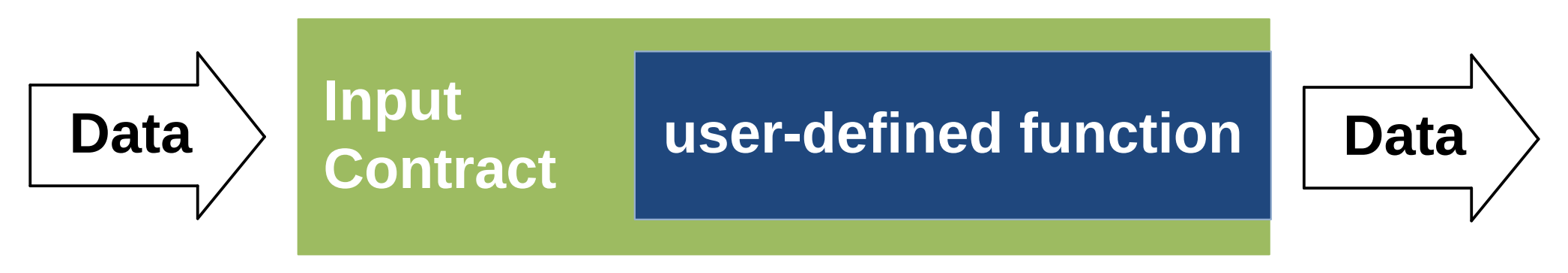

Fig. 1: Design of a Parallelization Contract (PACT).

Two PACT examples are shown in Figure 2. The input in the example consist of records with two attributes, each shown as a square. The first attribute is the key in our example. Equal box colors indicate equal key values. The Map function applies the UDF f to each input record independently. The Reduce function groups all records having the same key and calls the UDF g for each key-group once.

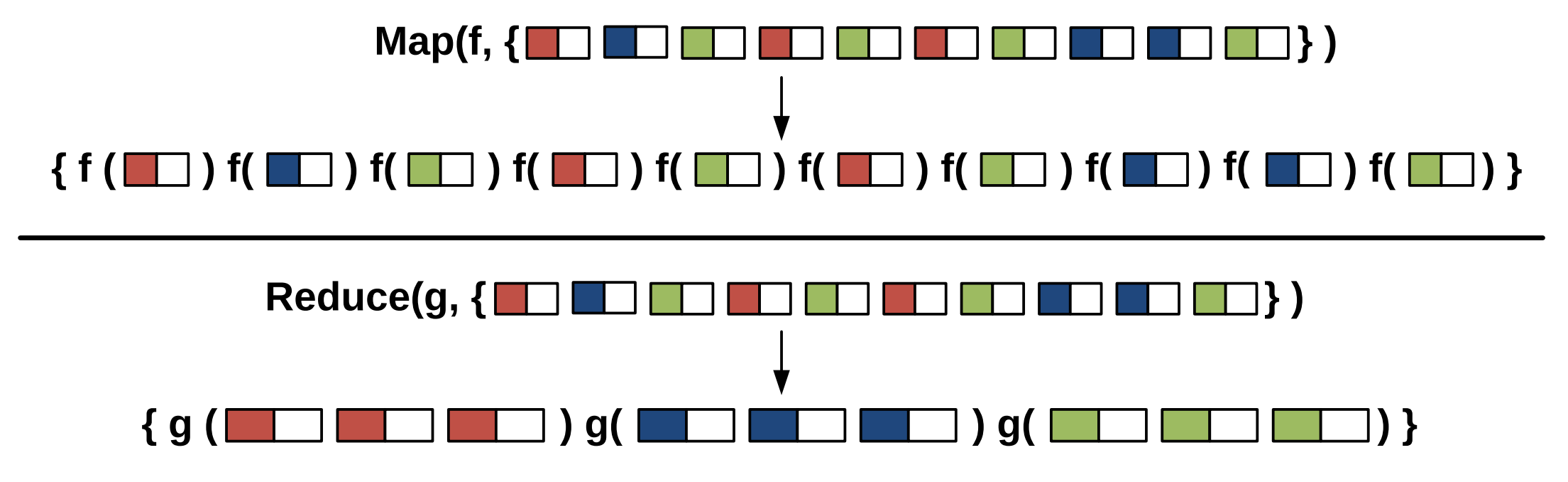

Fig. 2: Example for Map and Reduce.

Given a PACT program (left DAG in Figure 3) the input contracts define how the operators can be parallelized. Figure 3 shows an example with three operators A, B, and C. All three operators are parallelized with a degree of 4 (right DAG in Figure 3). In our example two different connection patterns are used. The instances of operator B are connected point-to-point with the instances of operator A. In contrast, a bipartite connection pattern in used from all C instances to all A instances.

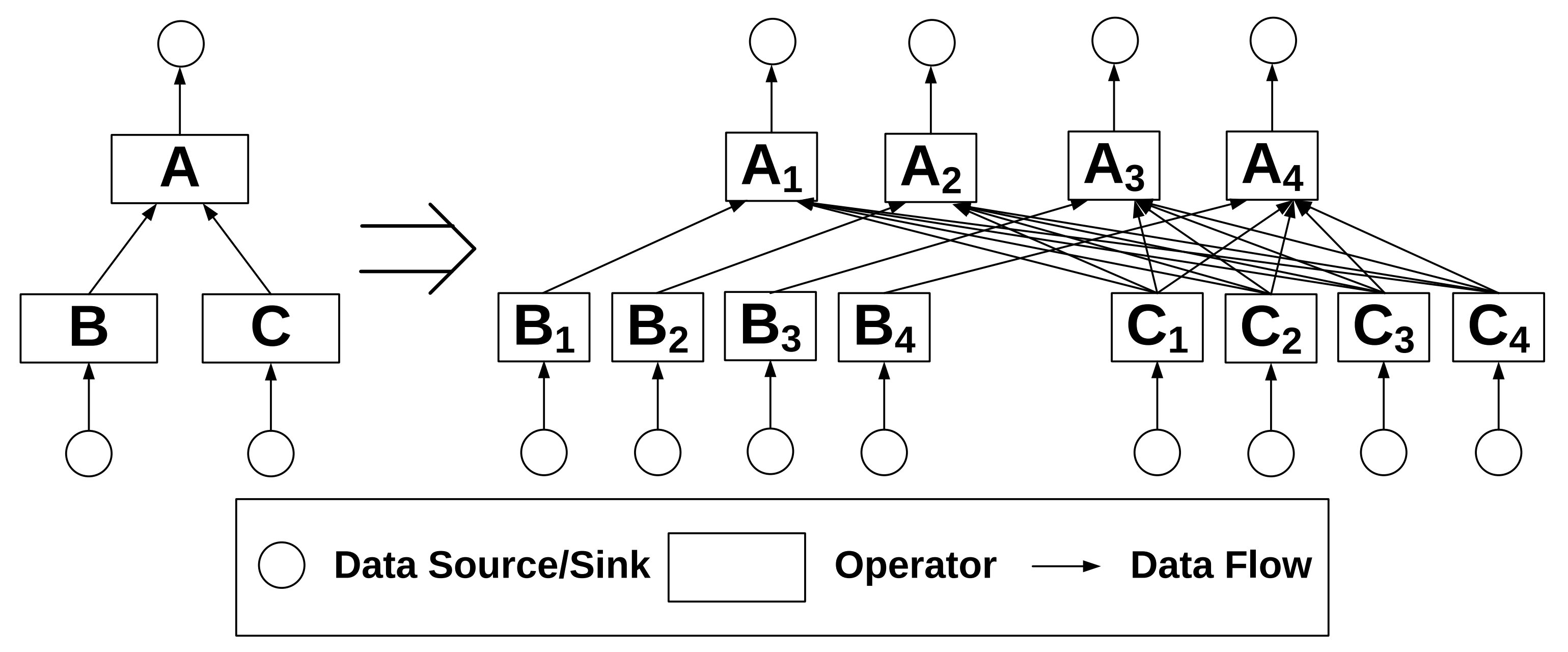

Fig. 3: Data-Flow Program and Parallelized Execution Graph.

Continuous Analysis of Sensor Data

Fig. 4: Wireless Network Nodes within the Humboldt Wireless Lab Equipped with Seismographic Sensors Collect Data.

The **Humboldt Wireless Lab (HWL)** testbed consists of about 120 wireless network nodes. Many of the nodes are equipped with sensors (e.g., seismographic sensors) for collecting data (Figure 4). Such a network might be installed as an earthquake early warning system within a city. As long as no earthquake occurs, the network infrastructure can be used to continuously collected sensor data. The deployment of multiple thousands of sensor nodes in a city results in gigabytes of collected data per minute.

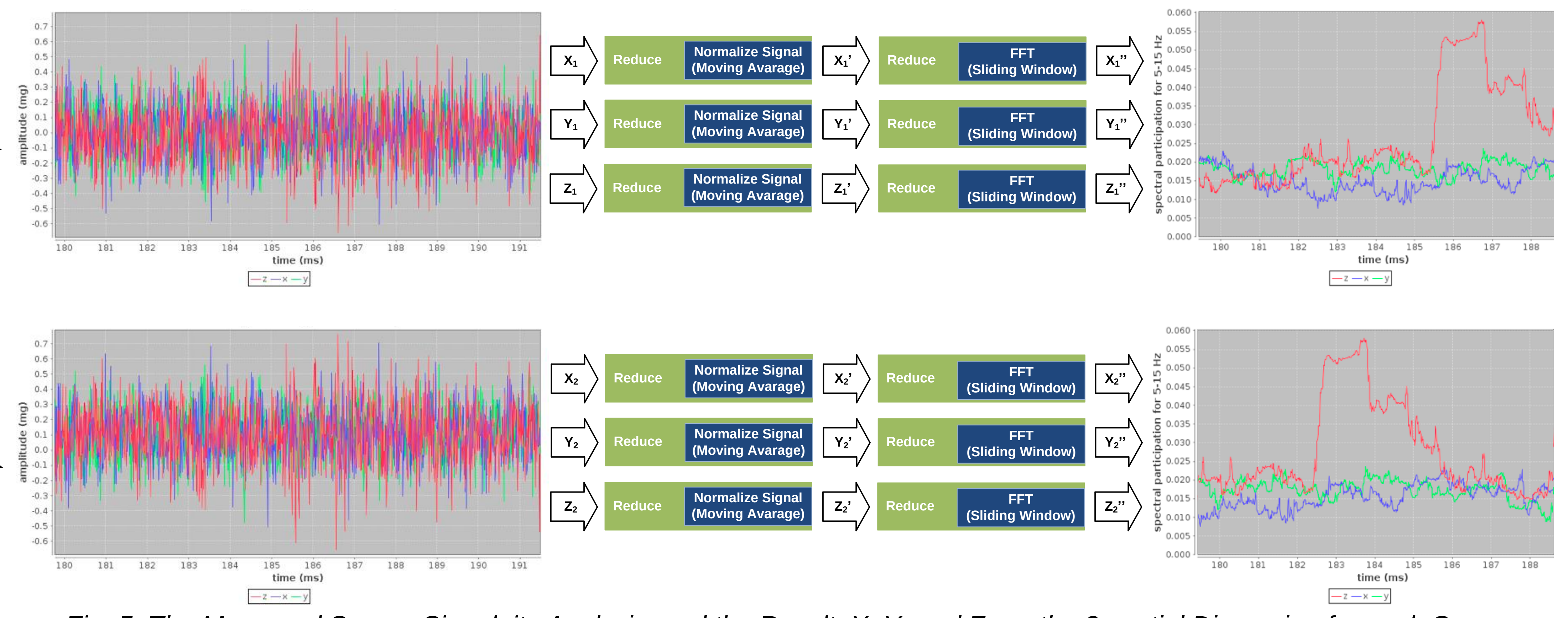

Fig. 5: The Measured Sensor Signal, its Analysis, and the Result. X, Y, and Z are the 3 spatial Dimension for each Sensor.

Challenges for Processing Sensor Data

Processing sensor data using a scalable parallel data-flow system like Stratosphere provides a good approach for process large amounts of data efficiently. However, Stratosphere (and similar systems) do not support the processing of infinite data streams such as the continues collected sensor data. Streaming systems are desigen for processing infinite input data. But those systems do not meet the following requirements:

- Existing parallel streaming system do not scale up to a large number of nodes and cannot deal with high data rates.
- At the same time, existing streaming systems are intended to be used for real time processing combined with low data processing latency. In our domain, we do not require real time constraints. Therefore we can relax the low latency restriction allowing for a simpler system design.

Example: "SeismoBus"

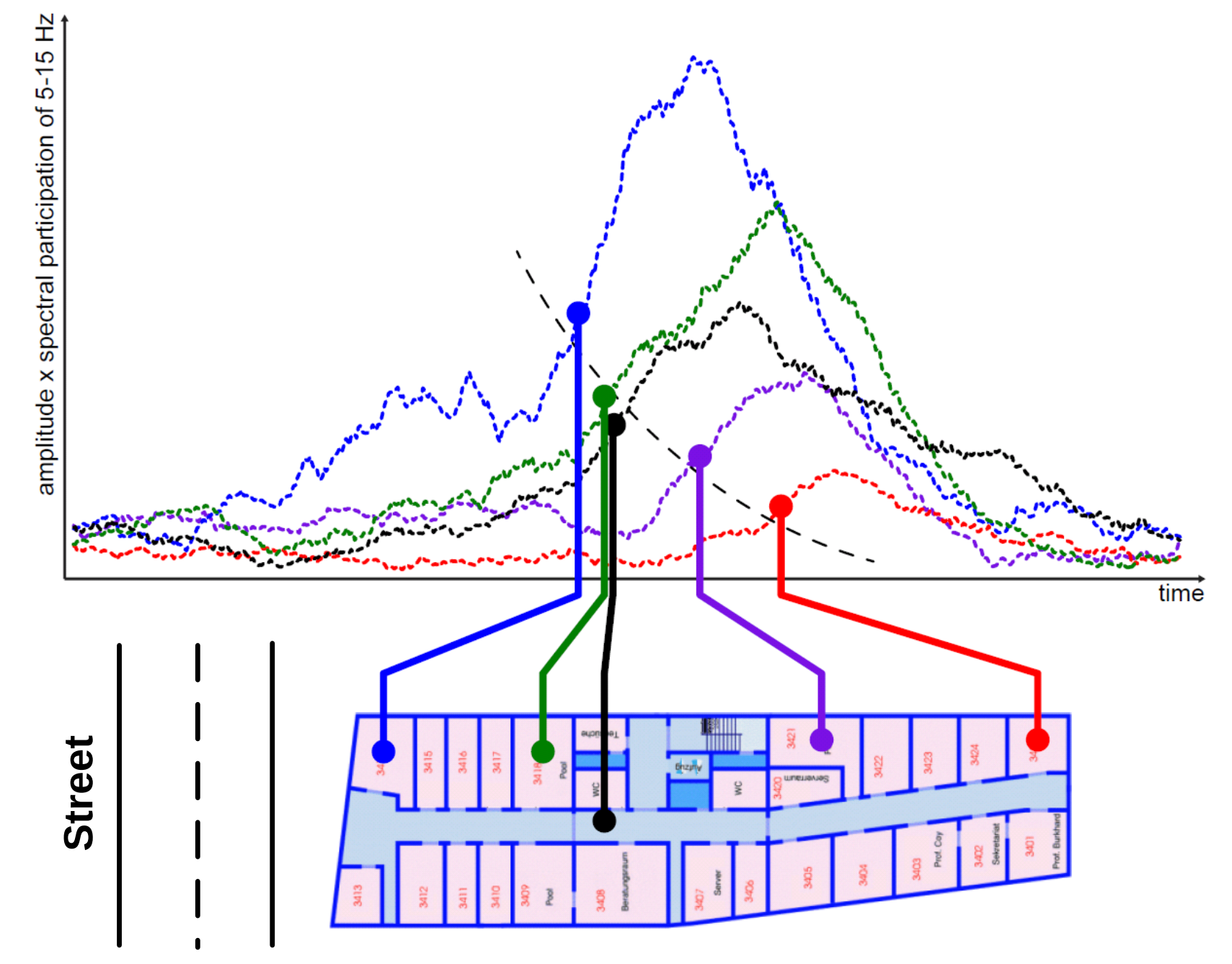

Fig. 6: Different Sensors with Damped and Delayed Amplitudes.

SeismoBus is an example for the analysis of seismographic sensor data. In this example, we detect a bus which passes by a checkpoint. Figure 5 illustrates the measured sensor signals of two sensors, the parallel analysis of the measured data using Fast Fourier Transformation (FFT), and the result of the analysis (in form of the transformed signals). The result of the analysis is aggregated and correlated with the location of the sensors. Figure 6 Illustrates the correlation: The amplitude of the blue sensor which is close to the street occurs first and has the strongest amplitude. All other amplitudes occur later and are damped because of their greater distance to the street.

Approach to Continuous Data Analysis

The goal of our work is to bridge the gap between MapReduce and streaming systems to support near real time processing of infinite data streams in a scalable parallel manner. We focus on building a scalable parallel and distributed system which can process continuously arriving input data. We are going to extend the MapReduce-like Stratosphere system for processing infinite input streams. Hence, we must define and implement new PACT operators having sliding window semantics. The choice of Stratosphere is based on two major reasons:

- In our sensor network scenario, we expect a highly unsteady system workload (i.e., day vs. night). Hence, we want to exploit the dynamic nature of a compute cloud by requesting and releasing virtual machines at runtime in order to scale our system up and down. To our knowledge, Stratosphere is the only system which is fully cloud aware and can adjust to the current workload automatically.
- Stratosphere's programming model PACT is an extension of the MapReduce model. Hence, writing more complex programs is easier and the system provides sophisticate query optimization.

Analysis of Flood Simulation Data

At the **German Research Centre for Geosciences (GFZ)** researchers run flood simulation in order to understand the flooding behavior of water after a dam broke. The flood simulation output are time series, each value in the time series is an matrix (called grid) describing the water level in a dedicated simulation area. Every simulation runs with a spatial resolution of 50 meters and contains about 800x1000 simulation points. Figure 7 shows the visualization of a grid at some point in time.

In order to investigate the input-output-correlation of the simulations, the output grids may be clustered. A naive clustering approach maps each grid into an high dimensional space and calculates the pair-wise distances of all grids using Euclidean distance metric. However, thus a calculation is very compute intensive and even massive parallel calculation does not result in an acceptable runtime. Hence, we develop new data mining methods for reducing the overall execution time. More precisely, we look for highly parallelizable methods that allows us to take advantage of parallel data-flow engines like Stratosphere.

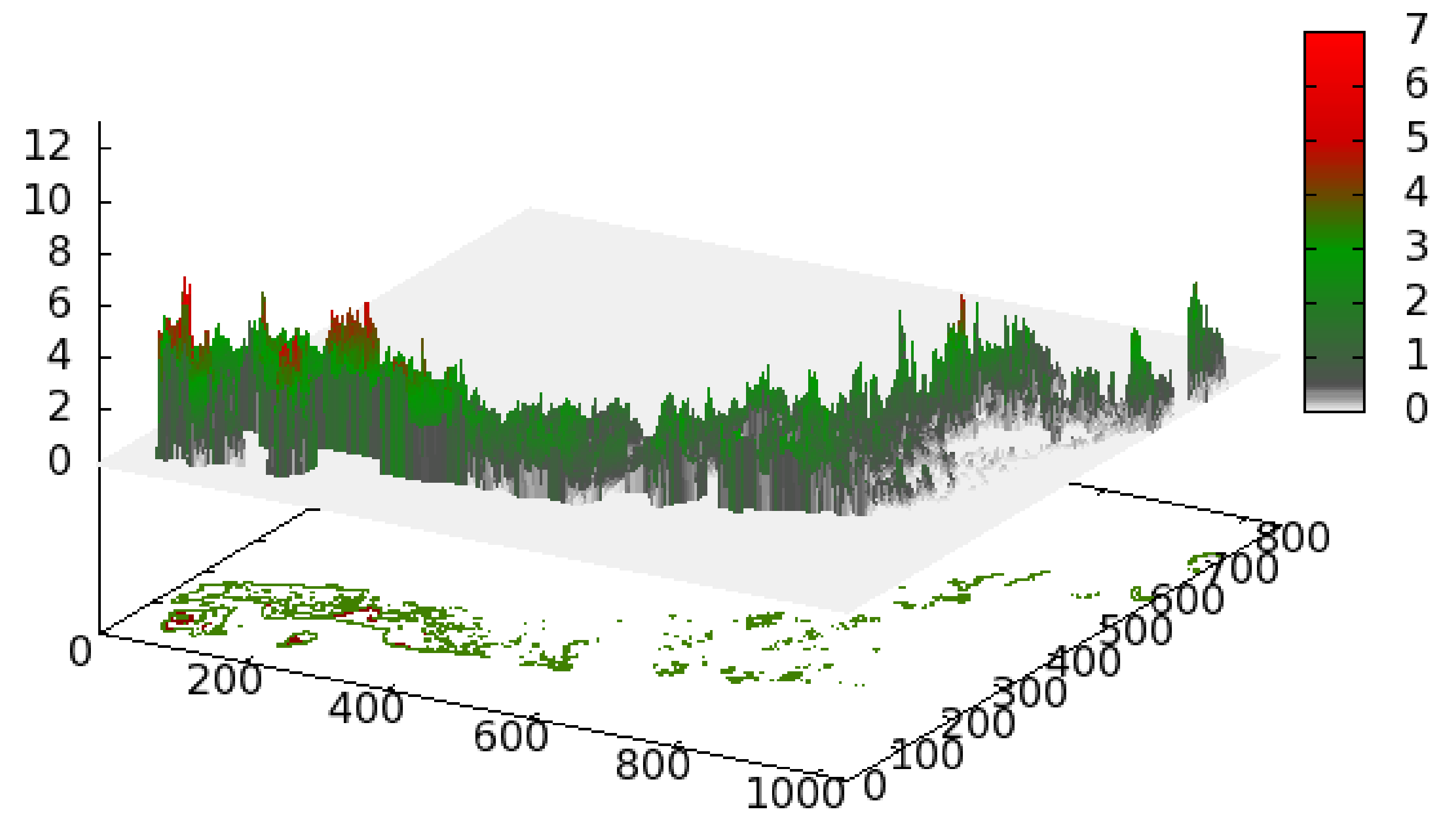

Fig. 7: Visualization of a Single Flood Simulation Grid (Water Level at Each Simulation Point in Meters).