DiAl: Distributed Streaming Analytics Anywhere, Anytime

Ivo Santos
Microsoft Research ATL Europe
Rablstrasse 26
D-81669 Munich, Germany
ivo.santos@microsoft.com

Marcel Tilly
Microsoft Research ATL Europe
Rablstrasse 26
D-81669 Munich, Germany
marcel.tilly@microsoft.com

Badrish Chandramouli
Microsoft Research
One Microsoft Way
Redmond, WA, 98052, USA
badrishc@microsoft.com

Jonathan Goldstein
Microsoft Research
One Microsoft Way
Redmond, WA, 98052, USA
jongold@microsoft.com

ABSTRACT

Connected devices are expected to grow to 50 billion in 2020. Through our industrial partners and their use cases, we validated the importance of inflight data processing to produce results with low latency, in particular local and global data analytics capabilities. In order to cope with the scalability challenges posed by distributed streaming analytics scenarios, we propose two new technologies: (1) JStreams, a low footprint and efficient JavaScript complex event processing engine supporting local analytics on heterogeneous devices and (2) DiAlM, a distributed analytics management service that leverages cloud-edge evolving topologies. In the demonstration, based on a real manufacturing use case, we walk through a situation where operators supervise manufacturing equipment through global analytics, and drill down into alarm cases on the factory floor by locally inspecting the data generated by the manufacturing equipment.

1. INTRODUCTION

Today there are 5 billion connected devices, by 2015 we will have 15 billion devices and the forecast for 2020 is 50 billion devices [1] . The growing demand to process and analyze the data coming from these devices requires new thinking and new strategies to handle effectively widely distributed data streams, especially when near real-time insights on the data are required. There is clearly not enough bandwidth to transfer all the raw data generated by these 50 billion connected devices to the cloud for further processing. And even if networking technologies made it be possible much of that raw data would be considered noise (i.e. uninteresting), wasting network and cloud resources. Adopting a local-global analytics strategy where data is pre-processed, pre-filtered to remove noise and pre-aggregated as close to the source as possible can help overcome the bandwidth utilization challenges, empowering the cloud to handle more efficiently large-scale global analytics over the already enriched data streams coming from the devices.

Local streaming analytics capabilities can complement cloud-based streaming analytics platforms and are typically very useful to process data born at the devices (e.g. geo location, device telemetry, asset specific data, sensor readings etc.), enabling data reduction (noise removal), as a mechanism to handle devices with occasional/intermittent connectivity patterns and in some scenarios even as a privacy preserving mechanism. In many domains, the user requires also a global view of his assets, i.e. his fleet, the status of his deliveries, or all the alarms in a refinery.

In a typical scenario (e.g. within an oil refinery or a manufacturing shop floor), sensors are connected to programmable logic controllers (PLCs) [2], where each PLC can be responsible for monitoring and controlling one machine. The PLCs forward information to a backend system where the signals can be correlated with manufacturing workflow data or enterprise resource planning. The operators' dashboards are reading and presenting the data from the backend system. Typically, there are already some static pre-filtering or down sampling rules on the PLCs (for example in normal situations, the PLCs should forward the temperature, vibration and rotation aggregated readings only every second). However, the PLCs have access to much higher data rates (up to 10 KHz). Suppose then that the PLC detects that the temperature is above a given pre-defined threshold and reports an alarm to the backend. In this situation, the operator would like to get more information from the machine/PLC associated with the alarm. Therefore, he grabs his mobile device (e.g. tablet or smart phone) and goes to the shop floor to check the machine personally. In front of the failing machine, he would now (1) like to connect to the machine directly and (2) inspect the high-resolution data streams. The direct access to these streams facilitates the visual detection of outliers or spikes associated with the alarm. He can even go further and post different queries on the data directly using his mobile device processing capabilities. To enable this, the operator's mobile device must be able to run local queries and the system managing the analytics must be able to detect the new context of the operator (in front of machine) in order to re-arrange the setup of analytics, so that he, the operator, can receive the high-rate data streams from the PLCs. Our demonstration will show the aforementioned scenario, highlighting the value of enabling real-time event processing in a heterogeneous, distributed setup. The key technical contributions introduced in this work are a new low footprint complex event processing engine designed for heterogeneous devices (JStreams) and a cloud-based service responsible for managing analytics artifacts deployed across cloud-edge topologies (DiAlM).

The paper describes the distributed analytics architecture and components in section 2, section 3 provides a demonstration walkthrough and section 4 is the conclusion.

2. ARCHITECTURE

Our architecture for distributed analytics (DiAl) aims to boost the connected devices story for cloud-based services and to provide easy-to-use analytics on devices. The architecture consists of two major components: (1) an engine enabling low-latency data stream processing on devices and also in the cloud, called JStreams, and (2) a service managing analytics artifacts (e.g. queries) across all connected nodes, DiAlM (see Figure 1).

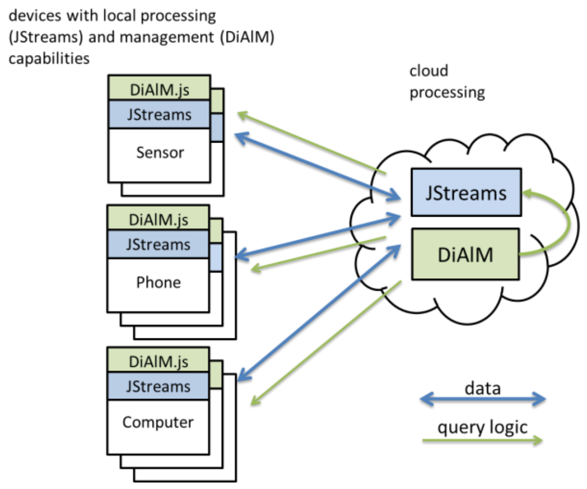


Figure 1: DiAl architecture

The DiAl architecture enables the realization of end-to-end scenarios where data born on devices, such as sensor or phones, can be locally pre-processed, sent to the cloud for further analysis with results then visualized also on edge devices, such as a tablets and smart phones

2.1 JStreams

JStreams is a complex event processing engine that implements the CEDR algebra [3] (like Microsoft StreamInsight [4]), with support for relational and temporal operations such as filters, projections, temporal joins, windowed aggregates, group and apply, lifetime manipulations etc.. JStreams is implemented in JavaScript and can run on practically any place where JavaScript runs, such as smart phones, tablets, web browsers, cars, or servers. JStreams is designed to be lightweight and cross-platform. It offers a simple but powerful programming experience, with an API designed especially for JavaScript developers. It also comes with built-in cloud connectivity to enable analytics manageability and streaming data connectivity from devices to cloud and cloud to devices in an easy way.

JStreams can run inside a device application container that connects physical sensors with inputs consumed by the queries. Some sensor values are exposed via HTML5/JavaScript APIs (e.g. location) and others can be accessed via platform specific connectors. The JavaScript runtime can be any kind of web browser, node.js [5] or a Windows 8 store application [6]. The results of the JStreams queries are sent to an output operator (a user defined function) that can, for example, display them in a local dashboard or send them to a service in the cloud (Figure 2).

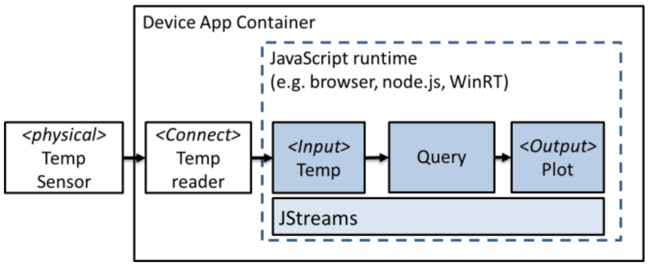


Figure 2: JStreams embedded in JavaScript container

Table 1 shows performance test results for some JStreams operators. Note that as with any JavaScript application only one core is actually used (no multithreading). The tests were executed using node.js [5] in an Intel Xeon W3530 PC (4 cores @ 2.67 GHz) with 10.0GB RAM.

Table 1. JStreams performance test results

Operator	Average throughput
Filter	2.0 Million events / sec
Project	1.7 Million events / sec
Min, Max, Average (temporal snapshot window)	0.5 Million events / sec
Min grouped by key (temporal snapshot window)	0.4 Million events / sec
Temporal Join	0.6 Million events / sec

2.2 DiAlM

The distributed analytics management service (DiAlM) coordinates analytics artifacts such as queries, inputs and outputs on distributed nodes (devices). The analytics are expressed as queries over continuous streams of data. The queries and device metadata are stored in the cloud (see Figure 3).

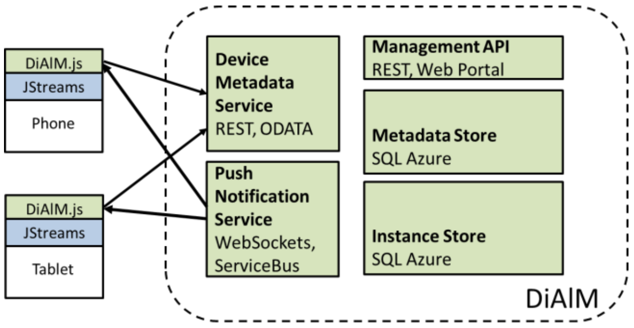


Figure 3: DiAlM platform

Devices interact with DiAlM via the device metadata endpoint, which offers a REST based API on top of ODATA [7]. The push notification service informs registered devices about new artifacts (e.g. new queries). The DiAlM service scales-out by design, as all state is stored in cloud-based distributed storage.

3. DEMO WALKTHROUGH

In our demonstration, we are showcasing a day in the life of an operator in an industry shop floor. A control room dashboard shows the process data of four machines (Figure 4). The PLCs, connected to the machines, collect high-resolution sensor data (sampling rates > 1000 Hz).

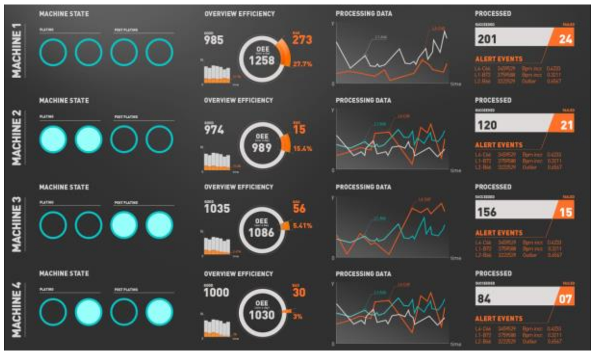


Figure 4: Control room dashboard

Sending all readings to the backend system is not an option (for bandwidth reasons), so the data is aggregated, pre-filtered and resampled directly on the PLCs using JStreams. On each PLC, we run two queries: (1) filtering alarm and (2) aggregating over a time window. Both queries are taking the power signal as input. The alarm filtering query checks the power input for power drops (where value < threshold) and the aggregation query calculates the average over a one second time window of the power signal. In the cloud (backend), we run a correlation query to get a global view of the energy consumption in our shop floor (see Figure 5).

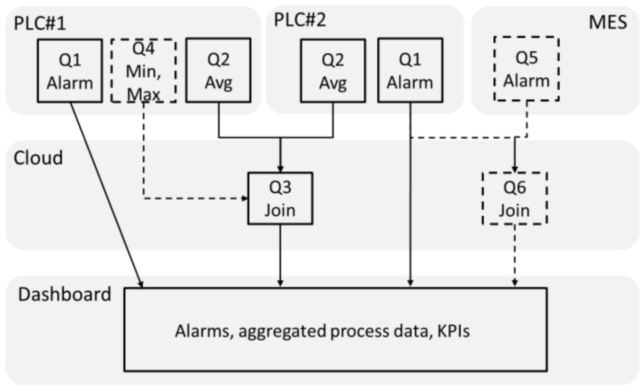
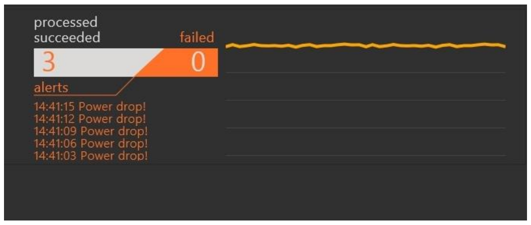


Figure 5: query plan

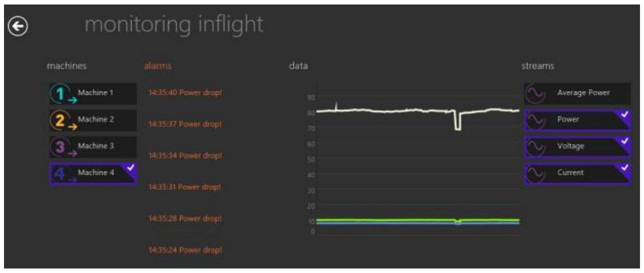
Suddenly several alarms occur for one of the machines, reporting power drops. An operator takes his tablet, opens the monitoring dashboard and observes the data coming from the machine: alarms and the actual down sampled signal (see Figure 6).



14:41:15 Power drop!
14:41:12 Power drop!
14:41:09 Power drop!
14:41:06 Power drop!
14:41:03 Power drop!

Figure 6: Alarm monitoring dashboard

Unfortunately, the data rates sent from the PLC to the backend are so low that the power drops are not visible in the signal. The operator might even wonder if the alarm really exists and is intrigued to understand the causes. The operator starts his investigation by connecting his mobile device directly with the PLC reporting the alarms to get high-resolution data streams from the machine, as he believes the power drops are somehow hidden in the aggregated data. With the high data rate, he is able to see the power drops now in the power line and it becomes clear that these are generating the alarms (see Figure 7).



14:35:40 Power drop!
14:35:37 Power drop!
14:35:34 Power drop!
14:35:31 Power drop!
14:35:28 Power drop!
14:35:24 Power drop!

Figure 7: Alarm monitoring dashboard

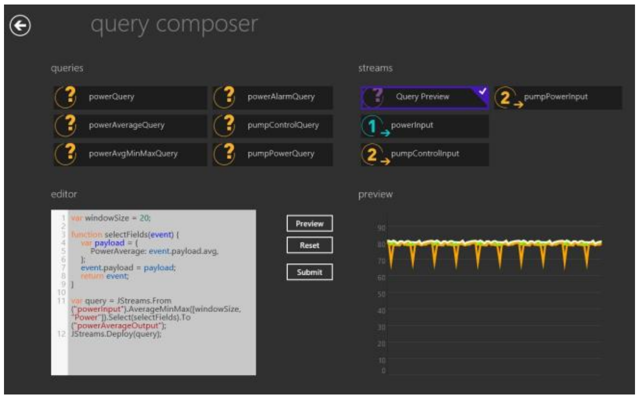
However, what is the root cause? To further investigate, he connects his mobile device to the manufacturing execution system (MES), what enables him to see all control commands submitted to the shop floor. His investigation leads him to a broken pump: whenever there is a power drop on his faulty machine, the pump turns on and at the same time the power signal of the pump shows an unexpected peak. The operator wants to use this discovered knowledge to enhance the dashboard output on the fly. Instead of just showing the aggregated power values, he wants to display the min and max values within a window as well and he wants to report enriched alarms that associate the power drop with the spike in the pump power signal.

He quickly drafts two new JStreams queries by downloading templates from DiAlM: query (1) aggregates over a time window of 1 second to calculate the min, max and average.

```
// Query 1

var minExpression =
     JStreams.From("powerConsumption")

    .Agg(“value”,
         “min”, JStreams.Min(),
         “max”, JStreams.Max(),
         “avg”, JStreams.Avg(),
         JStreams.OnHop(1, 1));
```

And query (2) issues an alarm whenever there is a power drop of the machine and a power peak for the pump with the information "*Power drop, please check pump!*".

```
var alarmPowerDrop =
    JStreams.From("powerConsumption")
    .Where(function (event) {
        event.value < threshold});

var alarmPumpPowerPeak =
JStreams.From("pumpPowerConsumption")
    .Where(function (event) {
        event.value > threshold});

JStreams.Join(
     alarmPowerDrop, alarmPumpPowerPeak)
    .Select("value",   "Power   drop,   please   check
pump!").Deploy("output");
```



```
var windowSize = 20;

function selectFields(event) {
    var payload = {
        PowerAverage: event.payload.avg,
    };
    event.payload = payload;
    return event;
}

var query = JStreams.From
("powerInput").AverageMinMax([windowSize,
"Power"]).Select(selectFields).To
("powerAverageOutput");
JStreams.Deploy(query);
```

Figure 8: Query composition and preview

The operator first previews the query execution directly on his device (Figure 8) and after verifying that it is producing the expected results he deploys query 1 to the PLCs and query 2 to the cloud (query 2 joins information from two different sources, PLC and MES).

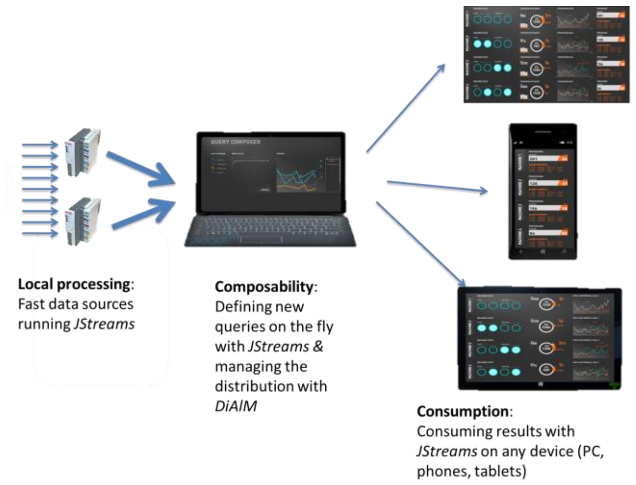

QUERY COMPOSER
Local processing:
Fast data sources running *JStreams*
Composability:
Defining new queries on the fly with *JStreams* & managing the distribution with *DiAlM*
Consumption:
Consuming results with *JStreams* on any device (PC, phones, tablets)

Figure 9: Demonstration flow overview

After deployment the enriched information gets visible on the dashboard within seconds. And the next time the pump breaks the operator also sees the new detailed alarm. Figure 9 summarizes the different demonstration steps.

4. CONCLUSION

This demonstration illustrates one of many scenarios where data born in connected devices can be leveraged in a distributed topology to gain business and operational insight. In a scenario inspired by a real-world use case, we show efficient asset monitoring in a manufacturing shop floor though local and global analytics, including drill down in alarm cases and root-cause analysis. We introduce the DiAl (Distributed Analytics architecture) and two main technologies: JStreams, a new complex event processing engine designed for heterogeneous devices and DiAlM, a cloud-based management service responsible for coordinating the different analytics artifacts (queries, devices, inputs, outputs etc.) across distributed cloud-edge topologies.

5. ACKNOWLEDGMENTS

We would like to thank Dirk Siemer, Christian Kratzsch, Louis Latour, Eldar Akchurin, and Rodrigo Alves for their help in building the demonstration and providing valuable feedback.

6. REFERENCES

[1] Dave Evans, "How the Next Evolution of the Internet Is Changing Everything," 2011. [Online]. Available: http://www.futuristspeaker.com/wp-content/uploads/Internet-of-Things-1.jpg.

[2] M. A. Laughton, D. J. Warne (ed), "Electrical Engineer's Reference book", 16th edition,Newnes, 2003 *Chapter 16 Programmable Controller*

[3] R. S. Barga, J. Goldstein, M. Ali, and M. Hong, "Consistent Streaming Through Time : A Vision for Event Stream Processing 2 . CEDR Temporal Stream Model," *General Systems*, pp. 363–374, 2007.

[4] Microsoft, "Microsoft StreamInsight," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ee362541.aspx.

[5] Joyent, "Node.js", 2013. [Online]. Available: http://www.nodejs.org/

[6] Microsoft, "Windows Store apps", 2013. [Online]. Available: http://msdn.microsoft.com/en-US/windows/apps/br229512

[7] Microsoft, "Open Data Protocol (OData)", 2013. [Online]. Available: http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-ODATA].pdf