PATH2iot: A Holistic, Distributed Stream Processing System

Peter Michalák, Paul Watson
School of Computing
Newcastle University
Newcastle upon Tyne, UK
Email: {P.Michalak1, Paul.Watson}@newcastle.ac.uk

***Abstract*—The *PATH2iot* open-source platform presents a new approach to stream processing for Internet of Things applications by automatically partitioning and deploying the computation over the available infrastructure (e.g. cloud, field gateways and sensors) in order to meet non-functional requirements including energy, performance and security. The user gives a high-level declarative description of computation in the form of Event Processing Language queries. These are compiled, optimised, and partitioned to meet the non-functional requirements using database system techniques and cost models extended to meet the needs of IoT analytics. The paper describes the *PATH2iot* system, illustrated by a real-world digital healthcare analytics example, with sensor battery life as the main non-functional requirement to be optimised. It shows that the tool can automatically partition and distribute the computation across a healthcare wearable, a mobile phone and the cloud - increasing the battery life of the smart watch by 453% when compared to other possible allocations. The *PATH2iot* system can therefore automatically bring the benefits of fog/edge computing to IoT applications.**

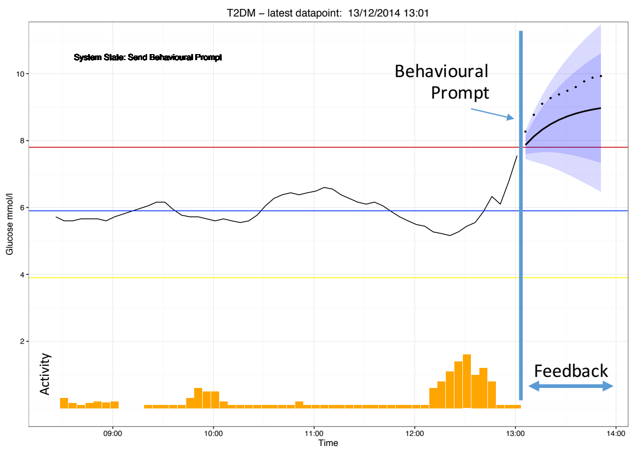

Fig. 1: Glucose and Activity stream processing and behavioural prompts with feedback.

I. Introduction

In this paper we present a new approach that simplifies the design and implementation of efficient systems for processing streaming data. IoT applications are becoming increasingly important in a wide variety of fields, including environmental monitoring, manufacturing and healthcare [1]. Extracting value from the data can be challenging, especially when it exhibits the high velocity and/or high volume commonly referred to as Big Data. In order to meet this challenge, stream processing engines have been created, especially for the cloud (e.g. Apache Spark[1] and Apache Storm[2]).

While these systems are highly efficient, this cloud-centric approach can present major problems for some important stream processing scenarios as they require data collected by sensors to be sent to the cloud for analysis. In particular, sensor battery life can be a major problem due to the energy cost of sending messages to the cloud. In this paper we describe how *PATH2iot* can address this problem, using a real medical application as the running example.

We are working with medical researchers on a healthcare application that uses wearable sensors to monitor the activity and glucose levels of type II diabetes patients and alert them before their health is endangered (Fig. 1). A Continuous Glucose Monitor periodically collects accurate glucose measurements from the patient. These must be analysed in order to give short-term forecasts: if patients' glucose levels are predicted to exceed the upper threshold for a healthy individual, a behavioural prompt (text message/notification) is issued to the user, asking them to heighten their physical activity in order to attenuate the upward trajectory of their glucose levels. Figure 1 presents an example of glucose and activity monitoring from this project, with an illustration of a behavioural prompt being issued based on the forecast glucose levels. When the type II diabetes patient recovers from the hyperglycaemic episode, the impact of their activity levels are estimated and feedback sent to the patient - a comparison between modelled (solid line) and actual glucose readings (dotted line). The impact of increased activity on glucose metabolic response is an active research area [2], [3].

To enable the incorporation of activity analysis into the model, the patient uses a healthcare wearable: a watch-like device that incorporates an accelerometer whose output can be processed to give a measure of their activity level.

The run-time analysis needed to achieve this cannot all be hosted on the wearable as it has limited computational capability, and battery life. Therefore, the wearable communicates data over the Bluetooth Low Energy local networking

[1]https://spark.apache.org/
[2]https://storm.apache.org/

protocol to a phone, which then sends the data over a mobile network to the Cloud for processing. If data is not partially processed in situ then every reading taken by the glucose and activity sensors must be transmitted to the cloud (via the phone) for analysis. As sending a message has an energy cost, this approach severely affects the battery life of the wearable, and the phone. If most of the messages are not required by the detailed analysis - for example, if only certain activity levels are of interest - then filtering out these messages close to the sensor could:

- dramatically increase battery life,
- reduce the number of messages transmitted over the mobile network (so reducing the required bandwidth),
- reduce the load on the cloud-based processing, which could be a serious issue if the application becomes popular, resulting in tens of millions of wearables streaming data to the cloud.

Further gains may be made by discarding unwanted data before it is sent to the cloud (e.g. the sub-fields of sensor readings that are not used in any computation), and by performing some basic analysis and data aggregation (e.g. averaging) on the wearable or phone.

In this paper, we describe and evaluate a system designed to enable this approach to stream processing in a way that reduces the complexity on the programmer. The *PATH2iot* - holistic, distributed stream processing framework automatically partitions and distributes processing across the available components (in this case wearable, phone and cloud) depending on the computational capabilities of the platforms, and the non-functional requirements (e.g. battery life).

The system also has the advantage that it automatically generates the software components that are deployed on each platform. This removes a major source of complexity from the programmer as each platform (e.g. cloud, phone and wearable) typically presents different software interfaces and challenges. To make this possible, the programmer specifies the stream processing computation in a high-level, platform-independent language (a set of Event Processing Language statements). An optimiser then determines how best to partition the computation defined by those statements across the available set of platforms in order to meet non-functional requirements such as energy in this case [4]. This takes into account the functional capabilities of the platforms (e.g. not all computations that can be performed on a cloud can be performed on a wearable or phone). *PATH2iot* can therefore be viewed as a way to automatically bring the advantages of fog/edge computing to IoT stream processing.

The rest of this paper is structured as follows. In Section II we present the overall system architecture and demonstrate its functionality with a healthcare, activity measurement example; Section III describes an example of optimising for a non-functional requirement: in this case using an energy-based cost model and how it is used during physical plan selection; Section IV is dedicated to an evaluation of the proposed approach, while the Discussion in Section V describes related work, and the future focus of the research.

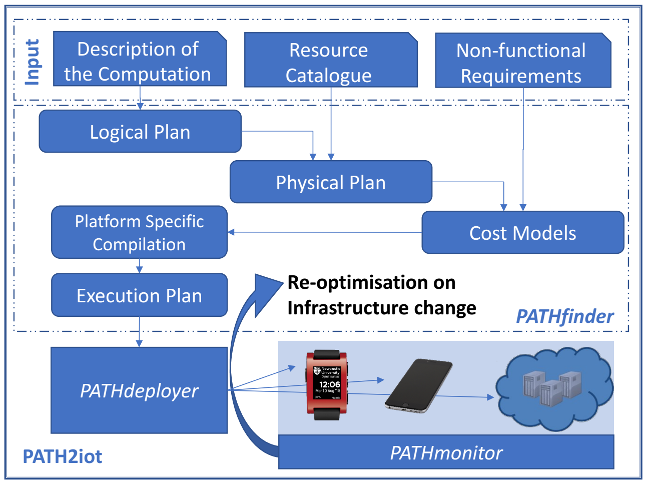
Input
Description of the Computation
Resource Catalogue
Non-functional Requirements
Logical Plan
Physical Plan
Platform Specific Compilation
Cost Models
Execution Plan
Re-optimisation on Infrastructure change
PATHfinder
PATHdeployer
PATHmonitor
PATH2iot

Fig. 2: The PATH2iot - System Architecture Overview.

II. System Overview

The overall system architecture is shown in Figure 2. It consists of three input components: the Resource Catalogue that describes the capabilities and characteristics of the available platforms (e.g. the wearable, phone and cloud), the Description of the Computation, and the Non-Functional requirements (e.g. energy). The optimiser (*PATHfinder*) takes these inputs and determines how to partition the computation over the platforms in order to meet the Non-Functional Requirements in the most optimal way. This output is passed to the deployer (*PATHdeployer*), which deploys it across IoT and Clouds. The monitoring module (*PATHmonitor*) will capture run-time information on the behaviour of the system, which can be used to re-optimise the computation if it detects that the non-functional requirements are not being met. We now describe each of these components in more detail, using the running example of Figure 1.

A. Description of the Computation

The ability to automatically partition a computation over a set of platforms requires that the computation be described in a high-level, declarative way that is amenable to analysis, distribution and optimisation. To meet these requirements, we adopted a relational model in which Event Processing Language (EPL) queries define the computation. This has three main advantages over alternatives. Firstly, SQL based languages, such as EPL, have been used to describe stream processing in a number of systems: Apache Spark [5], Apache Flink [6], and Esper [7]; this confirms that they are expressive enough for a wide range of applications. Secondly, EPLs are based on SQL, which is very familiar to a large portion of developers who will have used it to query databases. Thirdly, we can build on decades of work on optimising SQL queries in centralised systems [8], [9], [10], and in distributed query processing systems [11], [12] when designing the optimiser.

To illustrate the system we will use as a running example a part of a real digital healthcare application: a step count

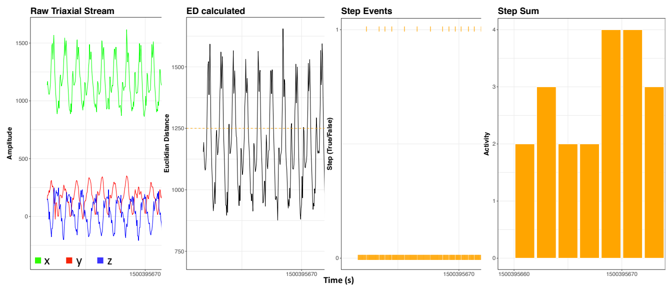

Fig. 3: The triaxial data stream from the watch transformed into an activity stream for the running example.

algorithm [13] that measures a users activity from information captured by the accelerometer in a healthcare wearable. Figure 3 shows the stages of processing: the raw triaxial data stream is transformed into the Euclidean Distance measure and a fixed threshold (THR) is applied. In the third stage, steps are detected, and then aggregated. This measure is used in the running example to evaluate patient activity levels.

The step count algorithm can be expressed through a set of EPL queries. These queries process a raw accelerometer data stream collected from a Pebble Steel Watch. The algorithm is defined using the following five EPL statements:

1) **INSERT INTO** *AccelEvent*
   SELECT getAccelData(25, 60)
   FROM *AccelEventSource*
2) **INSERT INTO** *EdEvent*
   SELECT Math.pow($x*x+y*y+z*z$, 0.5) **AS** ed, ts
   FROM *AccelEvent* **WHERE** vibe = 0
3) **INSERT INTO** StepEvent
   SELECT ed1('ts') as ts **FROM** *EdEvent*
   MATCH_RECOGNIZE (**MEASURES** A **AS** ed1, B **AS** ed2 **PATTERN** (A B) **DEFINE** A **AS** (A.ed $>$ THR), B **AS** (B.ed $\leq$ THR))
4) **INSERT INTO** *StepCount*
   SELECT count(*) **AS** steps
   FROM *StepEvent*.win:time_batch(120 sec)
5) **SELECT** persistResult(steps, 'step_sum', 'time_series')
   FROM *StepCount*

Query 1 includes a user-defined function (UDF) to read the data from the accelerometer. It has two parameters: the sampling frequency and the mode of operation. The mode of operation is 6 bits that determine which of the full set of possible data fields should be generated on the device, e.g. mode 0 does not generate any data, while mode 63 generates all data fields: x, y, z accelerometer axes data, vibe (state of the vibration module on the watch: 0 or 1, for active and inactive respectively), timestamp and battery level (expressed as a percentage, in a multiple of 10%). Mode 60, used in the running example, projects first four event fields (x,y,z, and vibe). Query 2 calculates the Euclidean distance from the raw accelerometer data stream for all of the data samples where the vibe parameter is set to 0. Query 3 processes the output of Query 2 and detects any spike crossing the specified threshold

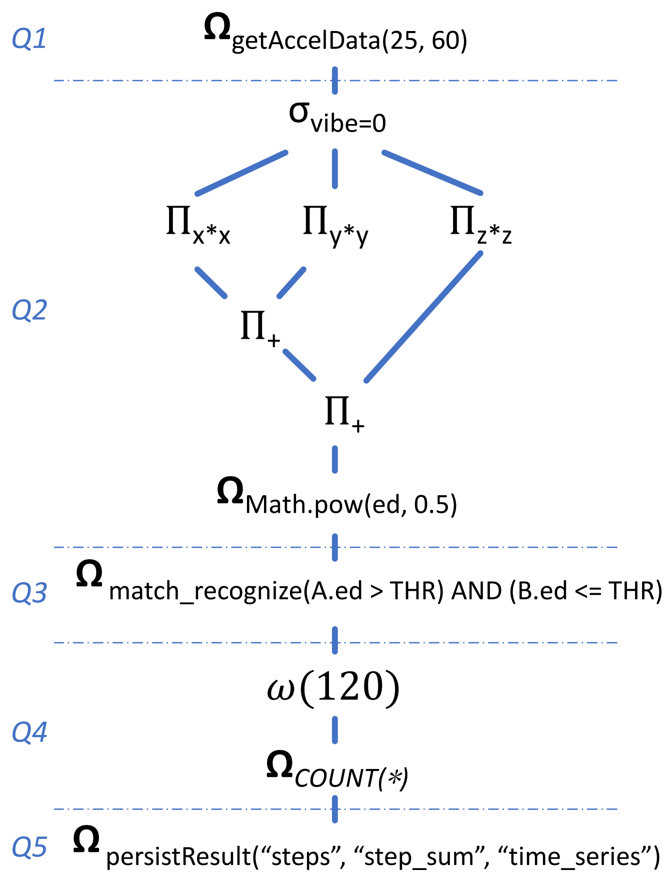
Q1
Ω getAccelData(25, 60)
σ vibe=0
Π x*x
Π y*y
Π z*z
Q2
Π +
Π +
Ω Math.pow(ed, 0.5)
Q3
Ω match_recognize(A.ed > THR) AND (B.ed <= THR)
ω(120)
Q4
Ω COUNT(*)
Q5
Ω persistResult("steps", "step_sum", "time_series")

Fig. 4: The EPL statements from the running example decomposed to a Computational Graph.

and forwards this information for further processing. Query 4 aggregates the input information and - based on a tumbling window regularly sends to Query 5 an event containing the number of steps taken. Query 5 then stores this in a database from where it can be accessed by healthcare applications for example to prompt the user with a text if action is needed to prevent a medical problem.

B. *Non-functional Requirements*

The system allows the application administrator to specify non-functional requirements such as performance, security, dependability and energy. The optimiser can then use this information to select the deployment plan that best meets these requirements. Our initial focus is on energy constrains; for the running examples, the aim is to maximise the number of hours the targeted battery-powered devices last between charges.

C. *Resource Catalogue*

The Resource Catalogue holds descriptions of the relevant features of the platforms over which the computation can be distributed. This includes the computational capabilities of all the platforms: for example, details of relational operators they can support - the optimiser accesses this information from the catalogue in the form of a JSON file that identifies all the platforms that can support a specific relational operator. In the running example, another key piece of information is the energy characteristics of all battery-powered platforms.

D. The PATHfinder: the optimiser module

The *PATHfinder* - a self-contained module within *PATH2iot* - takes a set of EPL queries as input, and determines how best to partition the computation over the set of platforms, taking into account the non-functional requirements and the capabilities of the platforms. It does this in the following main stages: EPL query decomposition followed by logical optimisation, and physical optimisation followed by device-specific compilation.

1) EPL query decomposition: utilises the native Esper SODA API [7] for EPL decomposition. The queries are loaded, decomposed and linked together - `INSERT INTO` clauses contain stream names that are used for this inter-query linking e.g. `INSERT INTO AccelEvent` from Query 1, links with `FROM AccelEvent` clause in Query 2.

2) Logical Optimisation: the set of EPL queries is decomposed into a computational graph of operators. Figure 4 shows this graph for the EPL queries in the running example. The description of operators is as follows:

- SELECT (σ): placing a constrain on events (such as vibe=0 in Query 2) filters out events from the stream propagated to downstream operators;
- PROJECT (Π): removing columns that are not needed from the events; and/or creating new columns through transforming existing ones (e.g. $x * x + y * y + z * z \rightarrow ed$ in the running example);
- Windows (ω): a variety of different window constructs can be expressed by [7]: tumbling, sliding, out-of-order correcting windows, etc.;
- User Defined Functions (Ω): the typical use is for generating new events (e.g. at a source node); persisting the events in a database (at a sink node). They are also used for all other use case specific computations that cannot be expressed using EPL statements. This allows arbitrary analysis computations, but should only be used where necessary as they limit the optimisation capabilities of *PATHfinder*, and narrow the placement options to those specific devices that can perform the given task.

The optimiser parses the query into a Logical Plan and uses stream optimisation techniques [14] including operator reordering to push windows closer to the data source, which can have a dramatic impact on the battery life of IoT devices (see Section IV).

3) Physical Optimisation: The operator graph generated by the logical optimiser serves as an input to the physical optimisation phase. The set of possible deployment plans is generated, and a cost model used to find the plan that best satisfies the non-functional criteria - in Section III we show in detail how one type of cost model, focussed on energy, is used to select a plan to deploy.

Device capabilities must also be considered at this stage: for example resource constrained devices do not have large amounts of RAM to store window data. The VersatilePebble smart watch application can hold up to 6000 Bytes of data within the program.

Another example of device capabilities that must be taken into consideration is when two platforms cannot directly communicate (e.g. the data from a Pebble Watch cannot be sent directly to the cloud as it transmits data over BLE. Instead an intermediate "relay" operator on the mobile phone has to propagate the data). To address this we adopted an approach similar to the data exchange service in [15]: a special *sxfer* operator is injected into the logical plan to enable for the data propagation.

4) Device-specific compilation: once the execution plan is derived, the tool translates the platform-agnostic operators into device specific code or configuration files. This information is send to *PATHdeployer* for IoT and Cloud deployment.

E. PATHdeployer: the deployment module

The holistic deployment strategy implemented within *PATHdeployer* consists of two stages: deployment in the cloud and IoT deployment. The deployment architecture overview was included in Figure 5.

1) Clouds: The *PATHdeployer* utilises several industry proven open-source technologies to deploy in clouds, such as Apache ZooKeeper - distributed coordination management system, and Apache ActiveMQ - scalable message broker; and D2Esper - developed in house as a dynamic real-time data stream processor based on Esper Complex Event Processing (CEP) library.

The *D2Esper* registers itself with a ZooKeeper node upon activation. Once a proper configuration set is delivered to the processor from *PATHdeployer*, it dynamically loads the event definition, parses the provided EPL statements and connects to a specified broker to start processing the real-time data stream. The output is forwarded to a specified destination (usually a different queue on the same broker) or a database. The tool utilises ZooKeeper's watcher capability to detect changes in a current configuration set and adapts stream processing accordingly. Automating deployment process, supporting heterogeneous platforms and enhancing computational dynamism with specific focus on IoT real-time stream processing is continuously improving with tools, such as [16].

2) IoT: To enable on-the-fly configuration of IoT devices the system has to rely on previously programmed capabilities, in this case on the phone and the wearable. This auto-configuration technique involves deploying an agent on each device. The agent periodically pulls and installs a configuration file from a REST API endpoint as shown on Figure 5. There are several advantages of this approach: being able to enact computation at a resource constrained device without the need for dynamic firmware update over the air - a capability not widely supported at edge devices, and an ability to change the computation during the runtime as well. The main disadvantage is that an agent offering a standardised approach for computational description and implementation must be designed and implemented for each IoT device.

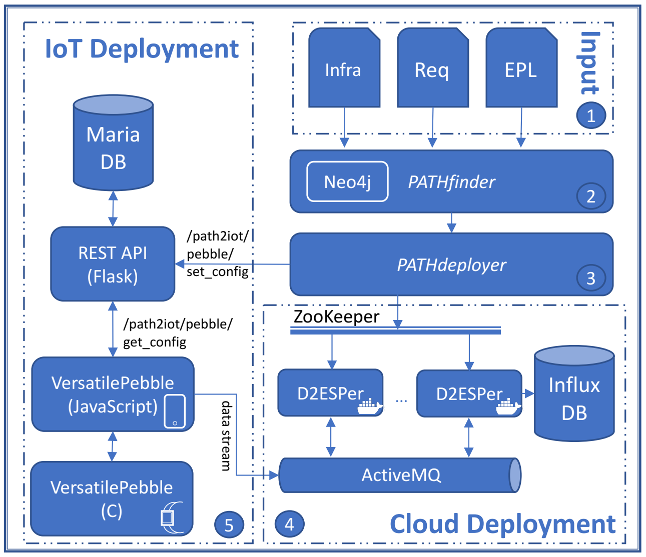

Fig. 5: IoT and Cloud Deployment Overview.

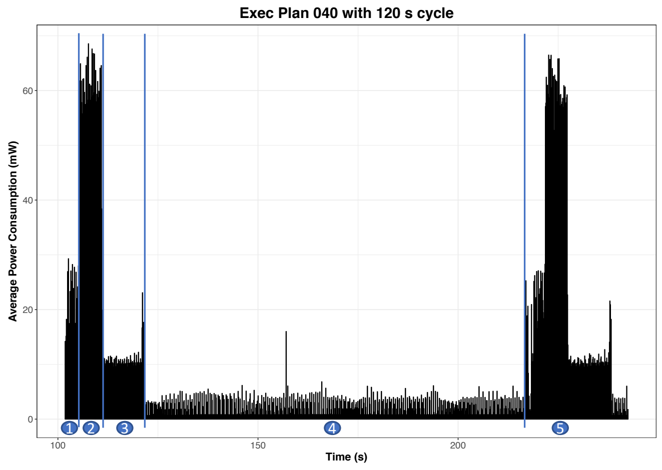

Fig. 6: BLE states as measured by Monsoon Power Monitor - single cycle (120s) view.

F. PATHmonitor: Real-Time Monitor

The real-time monitoring of IoT system is critical for any streaming system intending to adapt the computation based on the changes in the state of the infrastructure. Dynamic adaptation based on proactive re-optimisation triggered by the changes in the infrastructure is future work for *PATH2iot* [17].

III. An Energy Based Cost Model for Physical Plan Selection

The *PATH2iot* system allows the application administrator to set a minimum amount of time before the need to recharge the battery. For example: a doctor may want to guarantee 24 hours of battery life to ensure patients only need to recharge the wearable overnight.

Energy-aware management of IoT environments is of key importance within many other application areas, especially where replacement of a battery pack is infeasible or prohibitively costly [18]. To support the energy model [19], an experimental testbed was set up and calibration measurements made on real hardware.

The Energy Cost Model used by *PATHfinder* consists of three parts:

- Operating System (OS_{idle}) - a power consumption by the IoT device caused by the operating system. This is independent of any energy use by the application;[3]
- Computation ($comp\_cost$) - every computation in an application adds to the overall power consumption of the device. Each operation deployed on a node has an assigned device-specific energy coefficient derived from benchmarking tests, these are summed to obtain the overall impact of all operators on that node in a greater detail.
- Networking (net_cost) - networking has a higher impact on battery life than computation (see Table I). Therefore, messages transmitted between different nodes are considered in this formulae.

  A complex aspect of networking in the running example (and many others) is Bluetooth Low Energy Active State (BLE_{active}), that is activated every time a message is sent from the wearable device. It was observed that the OS keeps the BLE module active for a period of time in case there is another message to be sent. This is illustrated in Figure 6 - 1st phase: connection establishment; 2nd BLE messaging; 3rd $BLE_{activeState}$; 4th OS_{idle} with data generation and preprocessing, after which a cycle repeats (5th phase). The smart watch OS automatically activates a power saving mode after a certain period of time has passed, and it is important to express this in the energy model (this period was observed to be 10 seconds on the Pebble Steel device).

[3]It is important to note that the energy impact coefficients vary depending on the device and a version of operating system running on it.

An Energy-based cost model was developed to calculate the overall energy impact of an IoT node:

$$EI = OS_{idle} + \sum_{i}^{n} comp\_cost_i + \frac{msg\_count * net\_cost + BLE_{active} * BLE_{duration}}{cycle\_length} \tag{1}$$

The Energy Impact (EI) of all the deployment options was evaluated for each plan using this formulae. The OS_{idle} energy coefficient represents the impact of the operating system and the app running without any processing, network connectivity or data generation. The sum of all energy impact coefficients for each computations that is placed on the device is added up in the second component of the EI formula.

The last part of the EI formula estimates the impact of networking on the battery life of the targeted device. It is calculated based on the number of messages with a corresponding energy cost and a BLE_{active} component, which expresses the

TABLE I: Power Consumption Coefficients.

Operation	Energy Impact (mJ)	Conf Int
OS_{idle}	1.78	± 0.0370
25 Hz sampling	0.06	± 0.0153
SELECT	0.09	± 0.0416
ED	0.34	± 0.0665
POW	0.03	± 0.1039
WIN	0.06	± 0.0605
net_cost	5.06	± 0.2747
BLE_{active}	12.12	

power consumption of the Bluetooth module in standby mode. Figure 6 presents the power consumption for the execution plan with test ID 40. The first phase shows energy expenditure during a connection establishment; during second phase an active data transfer between the watch and a mobile phone is in process - sending preprocessed windowed activity events, the third phase has elevated power consumption due to the mentioned BLE standby mode - this impact on the overall battery life is not insignificant, which is why it must be included within the Energy Impact model. The fourth phase shows power consumption for the data generation, filtering, preprocessing, and windowing in an app memory. The 120 second cycle repeats at the phase 5 marker.

Table I presents power coefficients (rounded for brevity) calculated from the power measurement experiments for a set of computation and networking operations conducted using the Pebble Steel smart watch. We derived these coefficients by running a set of experiments, taking a mean measure of Energy used and subtracting other previously calculated impact factors with a calculated Confidence Intervals (see [20]).

Currently, we assume an infinite (or easily replenished) battery capacity on a mobile phone device (as experience shows that the wearable is the critical issue) and we only allow the application administrator to define non-functional requirements on the Pebble Steel watch.

IV. Evaluation

Key to the success of *PATHfinder* is the ability to correctly evaluate the cost model so as to select the best physical plan. This section describes how we derived the coefficients used by the energy-based cost model.

We benchmarked a Pebble Steel SmartWatch using a Monsoon Power Monitor[4]. This tool is a combination of a benchtop power supply and a measurement device with very high sampling frequency - up to 5,000 Hz.

The detailed results of the benchmark tests are summarised in a technical report containing all relevant data and code [20].

Table II presents results from the experimental tests monitoring the power consumption of the Pebble Steel Watch under different processing loads. A series of tests were designed and run to determine the Energy Impact of operations described in the graph of computation of a running example. The checkmark symbol under individual columns indicates that the given operator was active for that plan. An explanation for the columns within this table is:

- ID - an unique identifier for the test;
- Data - an operation to generate the triaxial data stream on a Pebble Watch with sampling frequency of 25 Hz;
- Select - filtering out any events that were collected while a vibration module on the watch was active, as in `WHERE vibe=0`;
- ED - transformation of raw triaxial data with the Euclidean distance calculation, reducing the amount of information propagating to the downstream operator $ED = x \times x + y \times y + z \times z$;
- POW - a Newtonian approximation of a square root (limited to 10 iterations), as the Pebble Watch doesn't have a Floating Point Unit on the chip;
- WIN - storing of the intermediary results inside a ring buffer: tumbling window of listed number of seconds;
- # msgs - number of messages send from the wearable watch to the phone within each cycle;
- Power (mW) - the average power consumed by the wearable under the current execution plan, as measured by the Power Monitor;
- EI (mJ) - the theoretical amount of energy to be consumed under the current execution plan based on the Energy Impact model;
- Conf Int - 95% confidence intervals calculated for the average cycle power measurements. We have selected this simplistic approach with assumptions of mutual independence of variables (operations) and approximately normal distributions for purpose of calculating these intervals. This is likely the reason for them being as narrow. However, they provide an indication of uncertainty of EI estimates.
- Bat (h) - an estimated battery lifetime based on EI result;
- Error (%) - a difference in estimated battery life based on measurements from Power Monitor and from EI model in percentile points.

To estimate battery life of a smart watch, an information of battery capacity is needed which was calculated using the following formulae: $maxBatteryLife = batteryCapacity \times batteryVoltage \times 3.6 = 130mAh \times 3.7V \times 3.6 = 1731.6J$.

The Figure 6 displays a power measurement capture from the Monsoon Power Monitor. The watch was programmed, as in test ID 040, to sample 25 Hz accelerometer data, select only events where 'vibe=0', calculated a square root of the Euclidean Distance, and window the output data stream within a 120 second tumbling window, then send the result to the mobile phone over BLE. It was observed, that the dominant energy impact on the battery comes from the networking.

If we compare a default scenario where the Pebble watch is programmed to stream all of the raw accelerometer data to the mobile phone (test ID 037 in Table II), so it can be relayed to the cloud for analysis, the expected battery life is 18.1 hours. In contrast, the optimised execution plan (test ID 060) has a power consumption of 5.88 mW, with estimated

[4]https://www.msoon.com/LabEquipment/PowerMonitor/

TABLE II: Execution Plans: Measurements and EI evaluation.

ID	Data Ω_1	SELECT σ_1	ED Ω_2	POW Ω_3	WIN ω_1	# msgs	Power (mW)	EI (mJ)	Conf Int	Bat (h)	Error (%)
037	✓	-	-	-	-	2.5	26.69	26.62	(26.48, 26.76)	18.1	0.24
060	✓	✓	✓	-	120	60	5.88	5.87	(5.70, 6.05)	81.9	0.06
040	✓	✓	✓	✓	120	60	5.97	5.91	(5.70, 6.11)	81.5	1.15
041	✓	✓	✓	✓	120	0	2.37	2.36	(2.22, 2.51)	203.5	0.11
042	✓	✓	✓	✓	-	0	2.31	2.30	(2.17, 2.44)	209.0	0.31
043	✓	✓	✓	-	-	0	2.27	2.27	(2.18, 2.36)	212.0	0.07
044	✓	✓	-	-	-	0	1.96	1.93	(1.88, 1.99)	248.8	1.25
045	✓	-	-	-	-	0	1.84	1.84	(1.80, 1.88)	261.2	0.04
046	-	-	-	-	-	0	1.78	1.78	(1.74, 1.82)	270.1	0.00
050	✓	✓	✓	✓	60	30	7.15	6.92	(6.71, 7.12)	69.6	3.34
051	✓	✓	✓	✓	30	15	9.63	8.93	(8.73, 9.14)	53.8	7.25
052	✓	✓	✓	✓	15	7.5	13.41	12.97	(12.77, 13.18)	37.1	3.27

battery life of 81.9 hours - an improvement of 453% with 95 % confidence intervals of 79.5 - 84.4 hours of estimated battery life, compared to 81.8 hours estimated by Monsoon Power Monitor (comfortably lying inside of the calculated intervals). Table III summarises other advantages of the optimised plan against the baseline including a data reduction factor of 3X - the amount of data in Bytes that is transmitted under the optimised plan is three times lower compared to the other.

As can be observed from comparison results in Table III, varying the length of window has significant impact on the battery life of the device. The longer the window, the less power the wearable device consumes. This is given by the fact that establishing the BLE connection, transmitting the data and keeping the BLE module active has a high energy impact on the battery. Sending the messages in rapid succession, and transmitting all the available data in short bursts reduces the overall power consumption. However, in the current software implementation, the maximum amount of the data that can be stored is 6000 Bytes, which places an upper limit of 120 seconds ($120s * 25 * 2B$) on the running example.

V. Discussion

A. Related Work

The recent interest in fog and edge computing - extending data processing into IoT environments - can be seen in research projects such as Linthicum [21] where the main aim is to reduce latency; in [22] which focusses on minimising the volume of data exchanged between fog and cloud nodes with a 'spillway' data structures; or [23] aimed at provisioning large-scale deployments within IoT environments. Also a trade-off between latency and cost has been investigated, such as in [24], where unit-slot optimisation techniques are used. Active interest in edge data analytics, such as in [25], [26] shows the need for in situ processing, especially for high velocity and volume data streams originating from IoT.

Recent work in [27], presents a bespoke solution for a dynamic computation offloading scheme, where an improvement of 21.1% in battery life was achieved by adjusting the partitioning of data processing between the wearable device and mobile application. The input from the application administrator was a classification accuracy for an audio-based nutrition monitoring use case. The proposed cost model, similarly to our Energy model, is based on the aggregate cost of extracting features, applying classification methods and transmitting necessary information between a phone and wearable device. However, in this work the cost models are derived and validated through simulation software without expressing uncertainty via confidence intervals, nor are there estimates for the duration of battery life of a wearable device.

In contrast, the *PATH2iot* takes a novel path of using a high-level declarative description of computation to automatically optimise and deploy across sensors, field gateways and clouds. As the tool is open-sourced other modules could be integrated to extend it capabilities for a specific scenario or additional non-functional requirements.

B. Future Work

Energy requirement is not the only possible basis for physical model selection process. We are also actively investigating possibilities to add other dimensions including Performance, Accuracy, Security, Monetary Cost and Dependability.

Future work on the healthcare application will explore adding and integrating more data streams and also expressing uncertainty for physical plans in more robust way. This includes adding the processing of continuous glucose readings as well as merging this with the activity information so as to provide a behavioural prompts for Type 2 Diabetes patients in near-real time.

A logical next step for *PATHfinder* is to consider the energy and performance trade-off: preserving battery life while keeping the latency at appropriate, user-defined levels. For example, in the running example, increasing the window size extends battery life, but increases the delay before information is sent to the downstream node. A more challenging future goal is the dynamic update to an execution plan when the monitor detects that the non-functional requirements are in danger of not being met.

VI. Conclusion

In this paper we have introduced a new system for holistically optimising and deploying IoT data analytics applications

TABLE III: Comparison between Baseline (ID 037) vs Optimised (ID 060) Execution plan.

Test ID	Computation	Power (mW)	Data TRF (kB)	BLE msg count (10 min)	EI (mJ)	Est Bat Life (h)
037	Ω_1	26.69	180	1500	26.62	18.0 - 18.2
060	$\Omega_1, \sigma_1, \Omega_2, \omega_1$	5.88	60	300	5.87	79.5 - 84.4

across heterogeneous platforms: e.g. sensors, field gateways and clouds.

The paper defined the overall system architecture comprising three main components: optimiser, deployer and monitor. An Energy based cost model with coefficients for a Pebble Steel smart watch was described and validated with the use of a Power Monitoring tool. Results showed that a dramatic improvement in battery life can occur when using the optimised execution plan rather than the baseline approach: this was 453% for the running healthcare example. This shows the potential for IoT management systems that can automatically exploit fog/edge computing to optimise non-functional requirements, including battery life for sensors. It has the additional advantage of being able to distribute stream processing computations across multiple platforms without the need for the application programmer to know how to program each type of platform: this is done automatically, by platform specific compilers and deployers, from a high level, declarative description of the computation. Future work will extend the range of cost models to include other criteria such as performance, and enable dynamic adaptation when non-functional requirements are threatened.

Acknowledgement

The authors would like to thank Dr. Matthew Forshaw for his expert advice on energy modelling, Dr. Sarah Heaps for statistics and Prof. Mike Trenell for the healthcare application.

This work is supported by the Engineering and Physical Sciences Research Council, Centre for Doctoral Training in Cloud Computing for Big Data (grant number EP/L015358/1) and Digital Catapult Teaching Grant Award (KH161118).

References

[1] S. Riazul Islam, D. Kwak, M. Humaun Kabir, M. Hossain, and K.-S. Kwak, "The internet of things for health care: a comprehensive survey," *Access, IEEE*, vol. 3, pp. 678–708, 2015.

[2] J. Henson, M. J. Davies, D. H. Bodicoat, C. L. Edwardson, J. M. Gill, D. J. Stensel, K. Tolfrey, D. W. Dunstan, K. Khunti, and T. Yates, "Breaking up prolonged sitting with standing or walking attenuates the postprandial metabolic response in postmenopausal women: a randomized acute study," *Diabetes care*, vol. 39, no. 1, pp. 130–138, 2016.

[3] S. R. Colberg, R. J. Sigal, J. E. Yardley, M. C. Riddell, D. W. Dunstan, P. C. Dempsey, E. S. Horton, K. Castorino, and D. F. Tate, "Physical activity/exercise and diabetes: a position statement of the american diabetes association," *Diabetes Care*, vol. 39, no. 11, 2016.

[4] P. Michalák, S. Heaps, M. Trenell, and P. Watson, "Automating computational placement in iot environments: doctoral symposium," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 434–437.

[5] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[6] Apache Flink, "Stream Processing for Everyone with SQL and Apache Flink," https://flink.apache.org/news/2016/05/24/stream-sql.html, [Online; accessed 19-July-2017].

[7] EsperTech, "Esper Reference - SODA API," http://esper.espertech.com/release-6.0.1/esper-reference/html/api.html#api-soda, [Online; accessed 19-July-2017].

[8] H. Garcia-Molina, *Database systems: the complete book*. Pearson Education India, 2008.

[9] J. Zhou, P.-A. Larson, and R. Chaiken, "Incorporating partitioning and parallel plans into the scope optimizer," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*. IEEE, 2010.

[10] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *ACM SIGMOD International Conference on Management of data*. ACM, 2013.

[11] D. Kossmann, "The state of the art in distributed query processing," *ACM Computing Surveys (CSUR)*, vol. 32, no. 4, pp. 422–469, 2000.

[12] J. Smith, P. Watson, A. Gounaris, N. W. Paton, A. A. Fernandes, and R. Sakellariou, "Distributed query processing on the grid," *The International Journal of High Performance Computing Applications*, vol. 17, no. 4, 2003.

[13] N. Zhao, "Full-featured pedometer design realized with 3-axis digital accelerometer," *Analog Dialogue*, vol. 44, no. 06, 2010.

[14] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A catalog of stream processing optimizations," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 46, 2014.

[15] P. Watson, "A multi-level security model for partitioning workflows over federated clouds," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, no. 1, p. 15, 2012.

[16] S. Mohamed, M. Forshaw, and N. Thomas, "Automatic generation of distributed run-time infrastructure for internet of things," in *Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on*. IEEE, 2017, pp. 100–107.

[17] T. Cooper, "Proactive scaling of distributed stream processing work flows using workload modelling: doctoral symposium," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM, 2016, pp. 410–413.

[18] M. Forshaw, N. Thomas, and A. S. McGough, "The case for energy-aware simulation and modelling of internet of things (iot)," *ACM ENERGY-SIM*, 2016.

[19] J. Allen, M. Forshaw, and N. Thomas, "Towards an extensible and scalable energy harvesting wireless sensor network simulation framework," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 2017, pp. 39–42.

[20] Peter Michalak, Matthew Forshaw, Paul Watson, "IoT Energy Monitoring Technical Report," https://github.com/PetoMichalak/iotPower, [Online; accessed 7-August-2017].

[21] D. S. Linthicum, "Connecting fog and cloud computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 18–20, 2017.

[22] M. Malensek, S. L. Pallickara, and S. Pallickara, "Hermes: Federating fog and cloud domains to support query evaluations in continuous sensing environments," *IEEE Cloud Computing*, vol. 4, no. 2, 2017.

[23] M. Vögler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "A scalable framework for provisioning large-scale iot deployments," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 2, p. 11, 2016.

[24] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, and A. Y. Zomaya, "Cost-effective processing for delay-sensitive applications in cloud of things systems," in *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*. IEEE, 2016, pp. 162–169.

[25] R. Fang, S. Pouyanfar, Y. Yang, S.-C. Chen, and S. S. Iyengar, "Computational Health Informatics in the Big Data Age: A Survey," *ACM Computing Surveys*, vol. 49, no. 1, Jun. 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2911992.2932707

[26] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, 2015.

[27] H. Kalantarian, C. Sideris, B. Mortazavi, N. Alshurafa, and M. Sarrafzadeh, "Dynamic computation offloading for low-power wearable health monitoring systems," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 3, 2017.