Multi-objective Deployment of Data Analysis Operations in Heterogeneous IoT Infrastructure

Devki Nandan Jha, IEEE Student Member, Peter Michalák, IEEE Student Member, Zhenyu Wen, Rajiv Ranjan, IEEE Senior Member, and Paul Watson

Abstract—The growth of Internet of Things (IoT) technology brings many new opportunities for applications in areas including smart healthcare, smart buildings and smart agriculture. These applications must normally distribute the computations, required for extracting value from sensor data, over the IoT infrastructure platforms (e.g. sensors, phones, field-gateways and clouds). This can be very challenging for IoT application developers due to the heterogeneity of aforementioned platforms, potentially-conflicting non-functional requirements (e.g. battery power, latency and cost), and related deployment criteria, which is impossible to resolve manually. To address the above challenges, we have developed the PATH2iot framework that decomposes a complex IoT application into self-contained micro-operations. Based on the deployment criteria, PATH2iot automatically distributes the set of micro-operations across IoT infrastructure platforms, while respecting their run-time data and control flow dependencies. In our previous work, we have shown how to use the PATH2iot to optimize the battery life of a healthcare wearable. In this paper, we describes a new research, that significantly extends PATH2iot, which introduces a heuristic model capable of making optimal deployment decisions based on multiple conflicting non-functional requirements and selection criteria (user preferences). It does so by leveraging a well known multi-criteria decision making method called the Analytic Hierarchical Processes (AHP). The applicability of the deployment model is validated based on a real-world digital healthcare analytics use case. The results show that our model is able to find the optimal deployment solution for different user preferences.

Index Terms—Internet of Things (IoT), Streaming Data, Analytic Hierarchical Process (AHP), Smart Healthcare

I. INTRODUCTION

Advances in IoT technology are transforming the society and the economy through their widespread impact, e.g. smart homes, smart healthcare and smart agriculture. This will continue to grow: research by Cisco predicts that 50 billion smart devices will be connected to IoT by 2020 [1]. To extract and process the massive streaming data collected from these data sources, several stream processing engines are developed that run in cloud providing a common programming frameworks e.g. Apache Storm [2], Amazon Kinesis [3].

However, this cloud-based approach is not suitable for many critical IoT applications for three main reasons. Firstly, some applications require close coupling between the IoT data generators and actions taken based on the analysis of the data [4]. For these applications the centralized cloud-based analytics approach might introduce unacceptable message transfer delays, and there may be a major problem due to network failure. Secondly, sending all the raw data from sensors to the cloud for analysis is not possible in cases where it may require higher bandwidth than is available or affordable [5]. Thirdly, sending all data to the cloud may flatten the battery of devices such as healthcare wearables too quickly [6].

To address these challenges, an alternative approach is to run part of the application on, or close to, the sensors that generate the data. In the literature, an IoT application can be represented using a set of queries (Q,s) which can be modelled as a directed acyclic graph (DAG) with data transformation operations (O(s)) as its nodes, and dataflow dependencies (or control flow dependencies for computational synchronization, if/then/else needed) between data transformation operations O₁ as its vertices (see Figure 1) [7]. This approach has been made possible by the introduction of what has become known as edge devices. Development of smarter IoT and edge devices, with some local storage and processing (e.g. healthcare wearables) opens up a tremendous opportunity for local analytics. Smartphones and field gateways can perform some basic analytic operations on the data, as well as acting as a network bridge between IoT devices and the cloud [8], [9].

Unfortunately, distributing applications across such a wide range of infrastructure (clouds, edge devices, sensors) (see Figure 1) is an extremely challenging task for a systems programmer, especially for critical IoT applications such as in healthcare and city management. Key challenges include:

- **heterogeneity of IoT infrastructure** - applications need to be deployed across a wide variety of heterogeneous platforms with differing programming interfaces and capabilities (e.g. sensors may have very limited computational capabilities). Data must be communicated between them over a variety of networks, each with its own idiosyncrasies and limitations.
- **meeting multiple, non-functional requirements** - optimizing several requirements is challenging as the requirements may be conflicting to each other e.g. performance, energy, cost, dependability.

These challenges leads to the following research questions:

**RQ 1.** How to model the problem of mapping complex IoT application across distributed IoT infrastructure with heterogeneous hardware and software configurations?
RQ 2. How to compute an optimal set of hardware and software configurations for each micro-operations of an IoT application considering conflicting non-functional requirements.

Early efforts centred on the deployment of IoT applications across cloud and edge datacentres are mostly theoretical. Moreover, these solutions have not considered automatic computation partitioning and deployment. Nor have they considered the optimization of multiple conflicting non-functional requirements during the deployment process.

In our previous work [6], we developed the PATH2iot system that allows an application administrator to define the overall computation in a high-level declarative Event Processing Language (EPL). The system decomposes the set of queries into a DAG that can be easily optimized. Next, it automatically partitions the applications into deployable operations, distributes them across the IoT infrastructure based on the battery power of the IoT device as the single non-functional requirement, and performs the physical deployment. However, this work did not focus on multiple non-functional requirements for making deployment decisions.

A. Contributions

To address the limitations of our previous work, this paper makes the following new contributions:

1) We provide a formal model for the deployment of a general IoT application across a distributed IoT infrastructure (e.g. sensor/wearable, edge/phone, cloud).
2) We prove that the partitioning and deployment of an IoT application across a distributed IoT infrastructure is a strong NP-hard problem.
3) We introduce a new heuristic model ABMO (AHP Based Multi-objective Optimization) based on Analytic Hierarchical Processes (AHP) [10] for finding the optimal deployment plan taking into account multiple, potentially-conflicting non-functional requirements. This includes user preferences for making decisions based on a set of non-functional requirements.
4) A comprehensive experimental evaluation is carried out using a real-world, digital health-care scenario for verifying the performance of the proposed decision-making technique.

The rest of this paper is organized as follows. Section II discuss the relevant related work while a formal model for the PATH2iot framework is presented in Section III. It also discuss the complexity analysis of the PATH2iot problem. Section IV describes the system model of PATH2iot, along with our proposed ABMO module. Section V evaluates the proposed framework on the real-world healthcare IoT application. Finally, Section VI concludes the paper and proposes some future work.

II. RELATED WORK

The problem of distributed stream processing has been extensively studied in the literature. Frameworks such as Stream [11], Flextream [12], Naiad [13] from academia and Apache Storm [2], Amazon Kinesis [3], Google MillWheel [14] from industry are common examples of stream processing systems. However, these approaches are usually limited to a cloud environment. Recent studies [8], [15], [16] show that data streams generated by IoT often requires local processing to satisfy different application requirements. Distributing the processing across sensors and edge devices along with the cloud introduces new problems mainly due to the heterogeneity of processing and storage capacity, the power status of the battery constrained devices, the mobility of the IoT devices and the interaction between them.

There are very few models that consider edge/fog and IoT devices for distributed stream application deployments. [17] proposes a framework for mobile fog computing that helps in the deployment of streaming IoT applications. [18] proposes a theoretical model for the fog computing infrastructure. It analyses the performance latency and energy performance of fog compared to the cloud environment. [19] proposes a programming infrastructure - “Foglets” - for distributing the deployment across edge and cloud. Foglets are used in the distributed deployment with latency and sensor mobility parameters taken into account. Benchmarks are used to test the application. Most of these models are theoretical and also considers only one or two parameters to be optimized.

In [20], an infrastructure module, LEONORE, is presented that provisions applications on resource-constrained edge devices. [21] presents the work that distributes the analytics of health monitoring application across edge processing. Similar work is done in Kea [22], a system for computational offloading of sensor data processing that also takes into consideration hardware capabilities, communication energy and latency costs. However, the focus of the system is limited to smart phone sensor data, compared to our system that views computational placement holistically.

A general model to support QoS-aware deployment is presented in [23], where a multi-component IoT application is deployed across fog infrastructure. The fog infrastructure here considers both edge and cloud environment. A simple Java-based prototype, FogTorch is presented to illustrate the proposed model. Although the deployment of an application across the IoT infrastructure is considered, it does not address how to optimize the deployment solution. Also, this is an abstract model that does not consider how to generate the distributed IoT application and how to perform the physical deployment. Recent work in [24] addresses the problem of dynamic computation offloading in wearable healthcare devices. An improvement of 21.1% in battery life is achieved by partitioning the computation between the wearable and a mobile device: the approach is derived and validated using simulation software.

To the best of our knowledge, this paper is the first to propose, implement and evaluate an optimized solution for distributed stream processing that considers how to optimize for heterogeneous non-functional requirements. Our framework, ABMO extends the capability of PATH2iot, is based on Analytic Hierarchical Process (AHP) [10] that incorporates user preferences along with a high-level computation description, non-functional requirements and a resource catalogue, in order to find an optimized deployment plan. AHP is a well known multi-criteria decision making method used in a variety of domains [25], [26].

III. FORMAL MODEL

In this section, we give some basic concepts and formally define our problem.

A. Basic Concepts

Definition 1. An IoT application $A$ is a triple $\langle DS, Q, \Gamma \rangle$ where
1. DS represents a continuous stream of data generated by the IoT device.

2. \( Q = \{ q_1, q_2, ..., q_k \} \) is the set of \( k \) queries defined by the user as a description of the computation. The set \( Q \) is logically decomposed using a stream optimization function \( \mathcal{P}(\cdot) \) into a set of computational micro-operations \( O \), which need to be deployed on the processing elements, as given in equation (1).

\[
O = \{ o_1, o_2, ..., o_l \} = \mathcal{P}(q_1, q_2, ..., q_k)
\] (1)

The dependency among various micro-operations is represented by a topologically ordered DAG. Each micro-operation \( o_i \) has specific hardware and software requirements \( R_H \) and \( R_S \) respectively. Some constraints \( Cons \) are also associated with the requirement specification.

3. \( \Gamma \) represents the identity property of the application and is represented as \( \langle id, r_H, r_S, cons \rangle \) where \( id \) is the identifier of application, \( r_H \) and \( r_S \) are the set of hardware and software requirements and \( cons \) is the set of constraints defined for the hardware/software requirement of the application.

**Definition 2.** An IoT infrastructure \( I \) is a quadruple \( \langle D, E, C, \lambda \rangle \) where,

1. \( D \) is the set of IoT devices \( d \), each represented by a set of 4-tuple \( \langle id, Type, S_H, S_S \rangle \) where \( id \) is the identifier. \( Type \) represents the type of device and \( S_H \) and \( S_S \) respectively represent the hardware and software support provided by the device \( d \).

2. \( E \) is the edge datacentre consisting set of edge devices \( e \), each denoted by \( \langle id, S_H, S_S \rangle \) where \( id \) is the identifier, \( S_H \) and \( S_S \) respectively represent the hardware and software support provided by the edge device \( e \).

3. \( C \) is the set of cloud datacentre components (VMs or containers) \( c \), each denoted by \( \langle id, S_H, S_S \rangle \) where \( id \) is the identifier, and \( S_H \) and \( S_S \) respectively represent the hardware and software support capabilities provided by \( c \).

4. \( \lambda \in \{ \{ D \times E \times C \} \cup \{ D \times E \} \cup \{ D \times C \} \cup \{ E \times C \} \} \) is a set of all the available connections from IoT device to edge to cloud.

**Definition 3.** The set \( R \) of non-functional requirements is a sequence of \( r \) elements \( R = \{ R_1, R_2, ..., R_r \} \) where each element \( R_i \) can have either numeric or boolean values. Unspecified values of element \( R_i \) are denoted by \( \theta \).

Numerous non-functional requirements may be associated with the application. Our approach is general, but those considered in this paper are discussed in Appendix A. Figure 2 shows the hierarchical representation for all the non-functional requirements considered in this paper.

**B. Problem Definition**

**Definition 4.** Let \( A = \langle DS, Q, \Gamma \rangle \) be an IoT application and \( I = \langle D, E, C, \lambda \rangle \) be the IoT infrastructure. A possible deployment plan \( \Delta_i \) is a mapping from operation \( O : \mathcal{P}(Q) \) to \( \lambda \in \{ \{ D \times E \times C \} \cup \{ D \times E \} \cup \{ D \times C \} \cup \{ E \times C \} \} \) (\( \Delta_i = O \rightarrow \lambda \)) if and only if:

1) all \( o_j \in O \) must be mapped to some IoT infrastructure \( I_k \in I \).

2) for each \( o_j \in O \), \( I_k \in I \), if \( (o_j \rightarrow I_k) \in \Delta \) then \( R_H(o_j) \leq S_H(I_k) \), \( R_S(o_j) \leq S_S(I_k) \) and satisfied(\( Cons(o_j) \)).

3) \( \sum_{j=1}^{max} R_H(o_j) \leq S_H(I_k) \) and \( \sum_{j=1}^{max} R_S(o_j) \leq S_S(I_k) \).

The definition given above considers all the constraints to meet the optional deployment requirements. Condition (1) guarantees that all the operations must be deployed on some IoT infrastructure. Condition (2) allocates the operations \( o_j \) only to infrastructure \( I_k \), which satisfies their hardware requirements \( R_H(o_j) \) and software requirements \( S_H(o_j) \), along with any other deployment constraints \( Cons \) defined for the operation \( o_j \).

Condition (3) limits the number of operations to be deployed on an infrastructure component so that the hardware and software requirements are enough to satisfy the demands of selected operations \( o_j \) for any element \( i \in \{ i, max \} \).

**Definition 5:** Given the available possible plans \( \Delta \), find the best possible plan \( \Delta_{best} \in \Delta \) that optimizes all non-functional requirements such that for any other plan \( \Delta_i \in \Delta \), \( \Delta_i \leq \Delta_{best} \) if and only if: \( \exists R_i \in R \) and \( \Delta_i < \Delta_{best} \).

**C. Complexity Analysis**

Given an IoT application \( A \) and IoT infrastructure \( I \), finding the optimal deployment plan \( \Delta_{best} \) from the set of possible plans

---

**Table I: A summary of symbols and abbreviations used in the paper**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>IoT application</td>
</tr>
<tr>
<td>DS</td>
<td>Data Stream</td>
</tr>
<tr>
<td>Q</td>
<td>Set of queries ( q_k )</td>
</tr>
<tr>
<td>( \Gamma )</td>
<td>Identity property of the application</td>
</tr>
<tr>
<td>O</td>
<td>Set of operations ( o_i )</td>
</tr>
<tr>
<td>( \mathcal{P}(\cdot) )</td>
<td>Stream optimization function</td>
</tr>
<tr>
<td>( R_H )</td>
<td>Hardware requirements of micro-operation ( o_i )</td>
</tr>
<tr>
<td>( R_S )</td>
<td>Software requirements of micro-operation ( o_i )</td>
</tr>
<tr>
<td>Cons</td>
<td>Constraints for micro-operation ( o_i )</td>
</tr>
<tr>
<td>id</td>
<td>Identifier of the component ( A )</td>
</tr>
<tr>
<td>( r_H )</td>
<td>Hardware requirements of application ( A )</td>
</tr>
<tr>
<td>( r_S )</td>
<td>Software requirements of application ( A )</td>
</tr>
<tr>
<td>cons</td>
<td>Constraints for application ( A )</td>
</tr>
<tr>
<td>I</td>
<td>IoT infrastructure</td>
</tr>
<tr>
<td>D</td>
<td>Set of IoT devices ( d )</td>
</tr>
<tr>
<td>E</td>
<td>Set of edge devices ( e )</td>
</tr>
<tr>
<td>C</td>
<td>Set of cloud datacentre components ( c )</td>
</tr>
<tr>
<td>( S_H )</td>
<td>Hardware support</td>
</tr>
<tr>
<td>( S_S )</td>
<td>Software support</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>Connection between different IoT components</td>
</tr>
<tr>
<td>R</td>
<td>Set of non-functional requirements ( R_i )</td>
</tr>
<tr>
<td>( \phi )</td>
<td>Unspecified values for ( R_i )</td>
</tr>
<tr>
<td>( \Delta )</td>
<td>Set of mappings ( \Delta_i ) for Operation ( O ) to ( \lambda )</td>
</tr>
<tr>
<td>( P_L )</td>
<td>Satisfied by</td>
</tr>
<tr>
<td>( P_{O,D} )</td>
<td>Set of logical plans</td>
</tr>
<tr>
<td>( P_{P,D} )</td>
<td>Set of all possible deployment plans</td>
</tr>
<tr>
<td>M</td>
<td>Comparison Matrix</td>
</tr>
<tr>
<td>CR</td>
<td>Consistency Ratio</td>
</tr>
<tr>
<td>CI</td>
<td>Consistency Index</td>
</tr>
<tr>
<td>RI</td>
<td>Random Index</td>
</tr>
<tr>
<td>( e_{ig} )</td>
<td>Maximum eigen Value of comparison matrix ( M )</td>
</tr>
<tr>
<td>( W_j )</td>
<td>Weight for non-functional requirement ( j )</td>
</tr>
<tr>
<td>N</td>
<td>Number of plans to shortlist</td>
</tr>
<tr>
<td>E1</td>
<td>Energy Impact</td>
</tr>
<tr>
<td>T^S</td>
<td>Total Turnaround Time</td>
</tr>
</tbody>
</table>

---

**Fig. 2: Non-functional requirements**
Fig. 3: Holistic representation of the deployment plan

\[ \Delta \text{ that optimizes all } R \text{ non-functional requirements is strong NP-hard and can be proved by reduction from the Bin-Packing problem.} \]

Bin-Packing is known to be a strong NP-hard problem, which is non-solvable in any polynomial time [27]. It is defined as: given a set of \( o \) objects with sizes \( s_1, s_2, \ldots, s_o \) and a set of bins \( B_1, B_2, B_3, \ldots \) of same capacities \( C \), find the smallest integer \( k \in N \) such that all the \( o \) objects got mapped to some bins \( B_i \) following the condition that for any \( i = \{1, 2, \ldots, k\}, \sum_{i \in B_i} s_i \leq C \).

**Proposition 1:** Finding an optimal mapping for PATH2iot problem in NP-hard.

**Proof:** Considering the formal definition of the Bin-packing problem as given above, it is possible to transform the Bin-Packing problem into simplest PATH2iot problem in a polynomial time. The transformation is as follows.

Change all the bins \( B_i \) to IoT infrastructure deployment nodes \( I \) (IoT device, edge device or cloud datacentre components) with equal hardware capacities and no software support. Change all the objects \( o \) to operations \( O \) with \( s_o \) hardware requirements, no software requirements, no constraints and no dependency between operations.

This maps the Bin-packing problem into the simplest case of our PATH2iot system deployment problem. This transformation can be easily achieved in polynomial time. Thus, proves that the simplest case of our problem is at least as hard as Bin-packing problem which is already strong NP-hard, making the generic PATH2iot problem \( \in \) strong NP-hard.

Inherently, as given in proposition 1, finding a solution of the Bin-packing problem in polynomial time leads to finding a solution of the generic PATH2iot problem in polynomial time. No such algorithm exists for any NP-hard problem, therefore we need a heuristic algorithm to find a solution.

### IV. SYSTEM MODEL

Figure 3 summarizes the internal processing of our proposed PATH2iot framework. The framework is divided in to three components. The first is the **User Input** that accepts a set of EPL queries which defines – in high-level terms - the computation, the state of the deployment infrastructure, and the non-functional requirements to be placed on the system. The second is the **PATHfinder** implementation, where all the deployment decisions are performed automatically, while the third is the **PATHdeployer** that performs the physical infrastructure deployment. **PATHfinder** is again divided into three stages, namely, **Initial Optimization**, **ABMO** and **Device Specific Compilation**. A detailed discussion of each component is given below.

#### A. User Input

The whole system is driven by the following inputs:

**Resource Catalogue.** It holds a description of all the relevant features of the IoT infrastructure platforms over which the computations can be distributed. This includes the infrastructure capabilities in terms of hardware support \( S_H \) and software support \( S_S \). It represents the state of the infrastructure, i.e. a description of the available cloud resources, IoT and Edge devices with their current state (e.g. active/disabled, battery level, battery capacity, network bandwidth). It also holds the constraints that need to be recognized when making deployment decisions. This includes a definition of User Defined Functions (UDF) that are supported by the system with their placement constraints, along with the Energy Impact coefficients for the supported operators. The optimizer accesses this information in the form of a JSON file.

**Computation Description.** To allow automatic partitioning over a set of platforms, it requires the computation to be defined in a high-level declarative language which can be analyzed, distributed and optimized. In this paper, we adopt the approach of [6] and use a Complex Event Processing (CEP) based relational model in which an Event Processing Language (EPL) is used to define the computation.

**Non-functional Requirements.** This defines all the requirements that are required to be optimized. The set of non-functional requirements considered in this paper are discussed in detail in Appendix A. The non-functional requirements are represented by a top-down hierarchical structure of level \( L \) where lower level elements are grouped under some higher level elements based on a common property, e.g. lower level attributes IoT cost, Edge cost and Cloud cost are grouped as they all calculate the cost.

**User Preferences.** This is one of the key user inputs as it defines the relative importance of the non-functional requirements. A user submits pairwise comparison values for all the non-functional requirements in the form of a CSV file.

#### B. PATHfinder

**PATHfinder** is an internal module of the PATH2iot system and is divided into three stages as explained below.

1) **Initial Optimization:** This stage is divided into two consecutive phases **Logical Optimization** and **Physical optimization**. The details are given below.

2) **Physical optimization.** This step generates a set of physical deployment plans based on the logical plans \( P_L \) generated by the previous step. For each logical plan, it first creates all possible deployment plans \( P_{OD} \) based on the topological ordering of the operations.

Algo 1 is used to shortlist the plans that satisfy the constraints and non-functional requirements and generates a list of physical deployment plans \( P_{PD} \).
2) AHP Based Multi-objective Optimization (ABM): This stage uses the Analytic Hierarchical Process (AHP) [10] for calculating the rank of each physical deployment plan. The whole process is divided into four steps as given below.

1) Calculating Weights. This step uses AHP to first find the weights for each non-functional requirement, based on the user-preferences. The preferences are measured by using Satty scale [29] as given in Table II. A reciprocal comparison matrix $M$ is constructed from the user-preferences following the rules as given in equation (2).

\[
M_{ij} = \begin{cases} 
1, & \text{when } i = j \\
X, & \text{when } i > j \\
1/M_{ij}, & \text{when } i < j
\end{cases} \tag{2}
\]

Before calculating the weights of the non-functional requirements, it is necessary to check whether the provided user-preference values are consistent or not. The consistency of comparison matrix is verified by checking the consistency ratio (CR) value. CR is calculated using the maximum eigen value ($eig$), size of the comparison matrix ($M.size$) and the Random Index ($RI$) values. The default $RI$ values are given in the Table III. Equation (3) is used to calculate the CR.

\[
CR = CI/RI \tag{3}
\]

where, $CI = (eig – M.size)/(M.size – 1)$. If the comparison matrix $M$ is found to be consistent, AHP is used to calculate the final weights $W_j$ for non-functional requirement $j$. Otherwise, the user is instructed to re-enter the preference values. AHP uses principal eigenvector to calculate the priority of each non-functional requirement. The final weights are computed by averaging the priority values. The pseudo code for the whole process is explained in Algo 2.

2) Normalization. Directly comparing different non-functional requirement values is not possible as each requirement can have a different data-type and range. Also, the optimal value depends on the type of requirements: in some cases higher is better e.g. Battery Power, while in others lower is better e.g. Cost. It is necessary to normalize all these values to one type and range. The normalization is performed according to the data type of the non-functional requirements e.g. boolean, numerical, etc. and the function whether to maximize or minimize. The steps of normalization process is summarized in Algo 3.

3) Plan Shortlisting. The complexity of the Pathfinder depends on the number of physical deployment plans. To manage the complexity of the Pathfinder, we shortlist $N$ plans from the full list of physical deployment plans. If the total number of the plans is less than $N$ we can skip this step.

Algo 4 explains the steps involved in the pruning process. It first ranks all the plans and finally selects top $N$ plans for further evaluation.

4) Multi-constrained Optimization. This step combines the final weights of the non-functional requirement with the corresponding values of the plans to find the final rank of the plans. This step aims to find the optimal plan over the selected plans. The rank of each plan $P_i$ is computed using equation (4), where $W_j$ is the weight for non-functional requirement $j$, and $\text{norm}(R_j(P_i))$ is the normalized value for the plan $P_i$. The plan with highest rank is selected for the physical execution.

\[
Rank(P_i) = \sum_{j=0}^{i} (W_j) \times \text{norm}(R_j(P_i)) \tag{4}
\]

3) Device-specific Compilation: Once the execution plan is derived, the framework converts the selected plan into a deployment configuration which is finally transmitted to PATHdeployer for deployment over the available IoT infrastructure.

C. PATHdeployer

There are two stages to deploy the optimized plan in PATH2iot: Cloud Deployment and Edge and IoT Deployment. The two stages are detailed below.

Cloud Deployment. is the first phase, where the tool verifies through the ZooKeeper that the destination D2SEMper instances are available in the infrastructure and then transmits the deployment configuration to them. The configuration information is then
Algo 3: Normalization

Input: $P_{PD}$ – list of physical deployment plans, $R_e$ – list of $r$ non-functional requirements, $type_{R_j}$ – data type of the non-functional requirement $R_j$, $Option_{R_j}$ – option for the non-functional requirement $R_j$ to be maximized or minimized, $R_j(P_i)$ – value of $j$th non-functional requirement for $i$th plan

Output: $norm(R_j(P_i))$ – normalized value of $j$th non-functional requirement for $i$th plan

1. initialize $norm(R_j(P_i))$ to 0 for all $R_j(P_i)$
2. for each $R_j \in R_e$ do
3. if ($type_{R_j}$ = Numeric) then
4. if ($Option_{R_j}$ = maximize) then
5. $Sum(R_j) = \sum R_j(P_i)$
6. $norm(R_j(P_i)) = R_j(P_i)/Sum(R_j)$
7. else
8. $Sum(R_j) = \sum norm(R_j(P_i))$
9. $norm(R_j(P_i)) = norm(R_j(P_i))/Sum(R_j)$
10. else if ($Option_{R_j}$ = minimize) then
11. $sum(R_j) = \sum norm(R_j(P_i))$
12. $norm(R_j(P_i)) = norm(R_j(P_i))/sum(R_j)$
13. else
14. option is not valid
15. end
16. end
17. if ($type_{R_j}$ = Boolean) then
18. if ($Option_{R_j}$ = maximize) then
19. $sum(R_j) = \sum R_j(P_i)$
20. $norm(R_j(P_i)) = R_j(P_i)/sum(R_j)$
21. else if ($Option_{R_j}$ = minimize) then
22. swap 0 with 1
23. do same as maximize
24. else
25. option is not valid
26. end
27. end
28. if ($type_{R_j}$ = Unordered_set) then
29. find the maximal set $R_{max}$
30. $norm(R_j(P_i)) = \text{size}(R_j(P_i) \cap R_{max})/\text{size}(R_{max})$
31. repeat the same process as in Numeric with maximize using the value $norm(R_j(P_i))$ in place of $(R_j(P_i))$
32. end
33. if ($type_{R_j}$ = Numeric_Range) then
34. find the optimal Range $R_{opt}$
35. $norm(R_j(P_i)) = \text{length}(R_j(P_i) \cap R_{opt})/\text{length}(R_{opt})$
36. repeat the same process as in Numeric with maximize using the value $norm(R_j(P_i))$ in place of $(R_j(P_i))$
37. end

Algo 4: Plan Shortlisting

Input: $P_{PD}$ – list of physical deployment plans, $R_e$ – list of $r$ non-functional requirements, $norm(R_j(P_i))$ – normalized value of $j$th non-functional requirement for $i$th plan, $N$ – maximum number of plans to be generated, $grade(P_i)$ – grade of the plan $P_i$

Output: $P_{PPD}$ – list of shortlisted physical deployment plans

1. initialize $grade(P_i)$ to 0 for all $P_i \in P_{PD}$
2. for each $P_i \in P_{PD}$ do
3. for each $R_j \in R_e$ do
4. $grade(P_i) = grade(P_i) + norm(R_j(P_i))$
5. end
6. end
7. sort $grade(P_i)$ in descending order
8. select top $N$ plans and store the list in $P_{PPD}$

parsed and dynamically compiled EPL statements are executed within the Esper CEP engine, which is wrapped inside the D2ESPer tool.

**Edge and IoT Deployment.** occurs once the cloud deployment has been completed. The configuration information is passed through a REST API endpoint. Pre-installed agents on IoT and Edge devices pull regularly configuration from the endpoint and once it has been received start the processing accordingly.

**D. Time Complexity of the Proposed Framework**

This section computes the time complexity of our proposed framework. Let $o$ be the number of operations and $T$ is the number of IoT infrastructure components. Consider the non-functional requirements $R$ is represented in a hierarchical structure with $L$ number of levels. The complexity of each phase is given below.

**Path Finder.** This phase is divided into three stages, and the complexity of each stage is given below.

a). Initial Optimization: There are two steps for this stage, the complexity of Logical optimization depends only on the operator reordering operation. For one Logical optimization, the maximum complexity of operator-reordering is $O(o)$. Varying the window size between a given range, $R1$ and $R2$ with defined step size $Step$ creates $((R1 – R2)/Step)$ plans. Since, $R1$, $R2$ and $Step$ is constant, the complexity for creating a set of logical plans, $P_L$ is $O(o)$. For Physical optimization, finding all the possible deployment plans for one logical plan takes $O(o \times T)$ searches.

Thus, the total complexity for finding all the optional deployment plan $P_{OD}$ for the given set of logical plan $P_L$ is $O(P_L \times o \times T)$. For finding the physical deployment plan $P_{PD}$, we have to check all the available possible deployment plan making the complexity as $O(P_{OD}) = P_L \times o \times T$.

After reduction, the overall complexity for Initial Optimization step is $O(P_L \times o \times T)$.

b). ABO: This stage consists of four steps, Calculating Weights, Normalization, Plan Shortlisting and Multi-constrained Optimization. Calculating weights compares the user preferences for each attribute by using matrix manipulation and computes eigen vector as the priority values. Given $n$ elements, the complexity to calculate the normalized eigen vector is $O(n^3)$. Considering $N_{lev}$ number of attributes at level $lev$ and $r_{sub,lev}$ number of sub-attributes at level $lev$ of subth attribute at level $(lev – 1)$, the complexity to calculate the normalize eigen vector is given as $O(\sum_{lev=1}^{L} \sum_{sub,lev=1}^{N_{lev}} r_{sub,lev})$. The complexity value seems large but is comparatively very small when compared with the comparison complexity without using AHP. The total complexity for comparing all the low level elements without using AHP is $(\sum_{sub,lev=1}^{N} r_{sub,lev})^3$ which is $>> (\sum_{lev=1}^{L} \sum_{sub,lev=1}^{N_{lev}} r_{sub,lev})^3$ when there are a large number of elements which can be represented in a hierarchical structure. AHP reduces the complexity by breaking the problem in a hierarchical structure resulting in more, smaller comparisons.

For the Normalization step, the time taken to normalize any non-functional requirement value is $O(P_{PD})$ irrespective of the data type. The complexity to normalize all $R$ non-functional requirements is $O(R \times P_{PD})$. Therefore, the final complexity for Normalization step is $O(R \times P_{PD})$. Plan shortlisting step finds the grade for each plan which is $O(R \times P_{PD})$ complex. The time taken to sort the grade values is $O((P_{PD})^2)$ and finally selecting the top $N$ plan is $O(1)$. Therefore, the final complexity for Plan Shortlisting is given as $O((P_{PD})^2)$. For final step, Multi-constrained optimization step, the complexity to calculate the rank for plan is $O(R)$. For all $N$ plan, the complexity is $O(R \times N)$ as $N$ is constant. Finally, the time taken to find the best value from sorted plan is $O(N) = O(1)$.

Thus the total time complexity for second stage is reduced to $O(\sum_{lev=1}^{L} \sum_{sub,lev=1}^{N_{lev}} r_{sub,lev})^3 + R \times P_{PD} + (P_{PD})^2$.

c). Device-specific Compilation: The complexity of this step is constant as there is only one execution plan for the deployment.
Path Deployer. The complexity of this step is also constant as the deployment is always performed for one selected plan.

Thus, the overall time complexity of our proposed framework is summarized as $O((P_L \times n \times \bar{T}) + \sum_{lev=1}^{L} \sum_{sub=1}^{N_{lev-1}} (r_{sub,lev})^3 + R \times P_{PD} + (P_{PD})^2)$.

V. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our proposed framework on a real testbed.

A. Experimental Setup

For experimentation, we choose a healthcare based application that captures the physical activity and blood glucose level of a type II diabetes patients [30]. The architecture is as shown in Figure 1. To analyze the activity, we used Pebble Steel smartwatch that contains an embedded accelerometer. The smartwatch connects to a LG G4 smart phone via a Bluetooth Low Energy (BLE) network and the phone is then connected to the cloud by via 4G mobile network. The data analysis uses a Step Count algorithm [31] that accepts the raw accelerometer data and generates activity information. Our framework - PATH2iot - automatically decides where to place the components of the analysis while optimizing different objectives (maximizing the battery life on the smartwatch and mobile phone, minimizing the turnaround time, etc.). The starting point is a description of the required computation as a set of EPL rules:

1) INSERT INTO AccelEvent
   SELECT getAccelData(25, 60)
   FROM AccelEventSource
2) INSERT INTO EdEvent
   SELECT Math.pow($x \times y + y \times z + z \times x$, 0.5) AS ed, ts
   FROM AccelEvent WHERE vibe = 0
3) INSERT INTO StepEvent
   SELECT * FROM EdEvent
   MATCH RECOGNIZE (MEASURES A AS ed1, B AS ed2 PATTERN
   (A B) DEFINE A AS (A.ed > THR), B AS (B.ed ≤ THR))
4) INSERT INTO StepCount
   SELECT count(*) AS steps
   FROM StepEvent.win:flexi_time_batch(30, 120, 15, sec)
5) SELECT persistResult(steps, 'step_sum', 'time_series') FROM StepCount

PATH2iot interprets this into a graph of basic operations, and then considers all the possible solutions that map these operations onto the available IoT infrastructure (Pebble watch, mobile phone and cloud). For each option, a cost is derived, and the one that has the maximal value of the non-functional requirements is selected.

In order to calculate the energy consumption of the Pebble Steel watch and the Mobile Phone, we must know the energy consumption of performing each operation (i.e. per sample received from the accelerometer). A series of experiments were conducted in which the watch and the mobile phone was connected to a Monsoon Power Monitor to measure the energy expenditure with each executed operation. Based on Central Limit Theorem (CLT), we assume that the entire data set follows Gaussian distribution. Therefore, we compute the 95% confidence interval ($Conf$) using equation (5), where $sd$ and $\bar{X}$ are the standard deviation and mean of the sample and $n$ is the size of the sample. The results are shown in Table IV and Table V. Notably, the Energy Impact shown in the two tables represents the mean of the sample.

$$Conf = \bar{X} \pm 1.96 \times \frac{sd}{\sqrt{n}} \tag{5}$$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy Impact (mJ)</th>
<th>95% Conf Int</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSIdle</td>
<td>1.78</td>
<td>± 0.0370</td>
</tr>
<tr>
<td>25 Hz sampling</td>
<td>0.06</td>
<td>± 0.0153</td>
</tr>
<tr>
<td>SELECT</td>
<td>0.09</td>
<td>± 0.0416</td>
</tr>
<tr>
<td>ED</td>
<td>0.34</td>
<td>± 0.0665</td>
</tr>
<tr>
<td>POW</td>
<td>0.03</td>
<td>± 0.1039</td>
</tr>
<tr>
<td>WIN</td>
<td>0.06</td>
<td>± 0.0665</td>
</tr>
<tr>
<td>netCost</td>
<td>5.16</td>
<td>± 0.2747</td>
</tr>
<tr>
<td>BLEactive</td>
<td>12.12</td>
<td>± 0.9516</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy Impact (mJ)</th>
<th>95% Conf Int</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSIdle</td>
<td>56.28</td>
<td>± 4.520</td>
</tr>
<tr>
<td>n_cost</td>
<td>161.62</td>
<td>± 6.813</td>
</tr>
<tr>
<td>RF_active</td>
<td>2497.10</td>
<td>± 226.288</td>
</tr>
</tbody>
</table>

To calculate the battery life resulting from each option, we need to know the overall capacity of the battery. The battery capacity and battery voltage of Pebble watch and LG G4 are 130 mAh, 3.7 V and 3000 mAh, 3.85 V respectively. The maxBatteryLife of Pebble watch and Mobile Phone $\beta_D$ and $\beta_E$ is calculated as $\beta_D = 130\text{mAh} \times 3.7V \times 3.6 = 1731.6J$ and $\beta_E = 3000\text{mAh} \times 3.85V \times 3.6 = 41580J$.

To calculate the total transmission time from Mobile phone to Cloud, we considered the Wi-Fi data rate from our phone (30 Mbps). Also, to calculate the Total Cost, we considered the standard electricity cost (ignoring the average fixed cost) as $0.155 / \text{KWh} [32]$ and standard data-rate cost as $0.01 / \text{MB} [33]$. We have neglected the VM cost for our experiment as the VM cost is almost same for all the cases. Different non-functional requirement values for our experiment can be found at [34].

B. Experimental Results and Analysis

We have compared three scenarios, where different non-functional requirements of the IoT infrastructure have been monitored. The detail of these scenarios are as follows:

1) Baseline. The generated raw accelerometer data (under 25 Hz) is streamed from the Pebble Steel watch to the cloud as quickly as possible. Given the software restrictions, this is done in a batch of 10 accelerometer samples, therefore with a frequency of 2.5 messages per second. This scenario gives the best Turnaround Time but consumes maximum energy.

2) Optimized by energy. The main focus, in this case, is to optimize the energy consumption of IoT device to increase the battery running hours. This is outlined in our previous paper [6], where, the optimizer selects the deployment plan where some of the operators are placed on the wearable watch, reducing the amount of data required to be transmitted and pushing windows closer to the data source in this case directly on the wearable device, with fixed window size of 120 seconds, greatly reducing the energy required to keep the Bluetooth connection opened. The result shows that we achieved a significant improvement in the energy consumption of the wearable device of 453% as compared to the Baseline approach. However, the Turnaround Time in this scenario is higher as compared to the Baseline approach.

3) Optimized by multiple conflicting objectives. Depending on the type of application and user requirements, the module
automatically selects the best deployment plan. It can be easily converted to the previous scenarios i.e. Baseline or Optimized by energy by setting up the user preference highest for Turnaround Time or Sustainability respectively. To illustrate the effectiveness of our proposed plan, we considered three cases with different user-preferences, as given below:

**Case 1:** In this case, the battery constraints are more important, so making the user-preferences inclined towards sustainability. The comparison matrix, $C_1$ for level 1 is given below. For other level, we considered equal priority for all the attributes.

$$
C_1 = \begin{bmatrix}
\text{Sus.} & T^3 & \text{Cost} \\
S_1 & 1 & 7 & 9 \\
T^3 & 1/7 & 1 & 2 \\
\text{Cost} & 1/9 & 1/2 & 1 \\
\end{bmatrix}
$$

**Case 2:** For this case, total turnaround time has higher preference as compared to other non-functional requirements. The comparison matrix, $C_1$ for this case is given below:

$$
C_1 = \begin{bmatrix}
\text{Sus.} & T^3 & \text{Cost} \\
S_1 & 1 & 1/7 & 2 \\
T^3 & 7 & 1 & 9 \\
\text{Cost} & 1/2 & 1/9 & 1 \\
\end{bmatrix}
$$

**Case 3:** In this case cost has given higher preference as compared to other non-functional requirements. The comparison matrix, $C_1$ for this case is shown below:

$$
C_1 = \begin{bmatrix}
\text{Sus.} & T^3 & \text{Cost} \\
S_1 & 1 & 2 & 1/7 \\
T^3 & 7/2 & 1 & 9 \\
\text{Cost} & 1/2 & 1/9 & 1 \\
\end{bmatrix}
$$

After completing the Initial Optimization stage, a total of 108 optimal plans got selected which acts as input for the ABMO. After execution of ABMO, the final result gives Plan107, Plan9 and Plan1 for the physical deployment in **Case 1**, **Case 2** and **Case 3** respectively. Figure 4 presents a clear comparison of the normalized values for all selected plans. The figure clearly shows that Plan1 has the best normalized value for communication cost and T3 with the value of (0.0157 and 0.0179) as compared to Plan9 and Plan107 with values of (0.0010 and 0.0082) and (0.0003 and 0.0045) respectively. However for remaining non-functional requirements, Plan9 and Plan107 gives better results. The detailed description of our implementation and result analysis is available at [35].

Based on the user preference values, the final rank is calculated as discussed in Section IV-B2. Depending on the users’ preferences, our proposed approach always select the optimal solution. Figure 5 shows the actual rank value for the best three plans. For **Case 1**, Plan107 has the highest rank value of 0.0345 followed by Plan9 with the rank value of 0.0261. For **Case 2**, the highest rank (0.0187) is shown for Plan9 followed by Plan107 (0.0158). Finally, for **Case 3**, Plan1 gives the best result (rank value of 0.0128) followed by Plan107 (rank value of 0.0124).

**VI. CONCLUSION**

PATH2iot provides a novel framework to facilitate the partitioning and deployment of IoT application across the distributed IoT infrastructure. This paper extends the basic PATH2iot framework by adding ABMO, a heuristic module leveraging AHP for making optimal decision based on users’ preferences and conflicting non-functional requirements. The case study shows that ABMO always chose the optimal deployment plan based on user preferences. The approach is very general; while we have described a healthcare use case in this paper, it can be directly applied to any other domain in which non-functional requirements are vital. For example we have other Smart City and transport applications, in which PATH2iot is used to minimize the bandwidth needed when transmitting data over a low-bandwidth network. The system also works well when a model derived by machine learning is used to classify, or predict behaviour: in this case, the model is simply treated as an operation in the computational graph, and PATH2iot is then used to decide where to place it in order to meet the non-functional requirements. In addition to this, our model is also compatible with the IoT environment that includes software defined networks (SDN). To interoperate with the SDN infrastructure, a query operation needs to be composed and deployed based on the NFV technique. Finally, whilst the focus of this paper is on stream processing IoT application, the approach can be easily applied to batch query processing workload. To this end, a batch query can be modelled as an operation, within our IoT graph, deployed at either edge or cloud layer. Since batch processing is normally executed over massive historical data, batch query operation are most likely to be mapped to the Cloud layer. However in the future work, we will make the deployment decision more dynamic so that if the environment changes over time, the deployment plans are automatically adjusted to maintain the non-functional requirements specified by the user.

**ACKNOWLEDGEMENT**

This work is partially 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).
APPENDIX A
NON-FUNCTIONAL REQUIREMENTS

The details of different non-functional requirements considered in this paper are as follows.

Sustainability. This considers the energy impact in terms of the battery life of all the IoT and edge devices. Battery life is very important to consider as it can be a limiting factor: for example, healthcare wearables that don’t last a full day on a single charge of the battery are not going to be used in practice. Battery power is represented in terms of Energy Impact (EI) which is measured in Milli Joule (mJ). As an example, the energy impact for a BLE connected IoT device \( EI_D \) and edge device \( EI_E \) is calculated as given in equation (6) and equation (7).

\[
EI_D = OS_{idle} + \sum_{i} c_{costi} + \frac{msg\_count_D \times n\_costD + BLE_{active} \times BLE_{dur}}{cycle\_length} \tag{6}
\]

\[
EI_E = OS_{idle} + \sum_{i} c_{costj} + \frac{msg\_count_E \times n\_costE + RF_{active}}{time\_slice} \tag{7}
\]

Where, \( OS_{idle} \) is the power consumption by the IoT device/edge device caused by the operating system, \( c_{cost} \) and \( n\_cost \) are the overall power consumption of the IoT device (D) and edge device (E) for each computation and transmission respectively, \( msg\_count \) specifies the number of messages transmitted from IoT device to edge device or edge device to cloud device, \( BLE_{active} \) is the power consumption for the Bluetooth low energy active state that is activated just after the message transfer, \( BLE_{dur} \) is the period of \( BLE_{active} \) state, \( cycle\_length \) is the time duration after which the whole cycle repeats and \( time\_slice \) is the particular period of time considered for edge device.

The energy impact value represents the average power consumed by the IoT device and the edge device. The battery life is inversely proportional to the energy impact. The battery life in hours \( bat_D \) for IoT device \( D \) is computed as given in equation (8).

\[
bat_D \propto (EI_D)^{-1} \Rightarrow bat_D = \beta_D \times (EI_D)^{-1} \tag{8}
\]

Where, \( \beta_D \) is a constant called \( maxBat \) that depends on IoT device \( D \) and is given by equation (9) where \( bat(C) \) is the battery capacity and \( bat(V) \) is the battery voltage.

\[
maxBat = bat(C) \times bat(V) \times 3.6 \tag{9}
\]

Similarly, for an edge device, the battery life in hours \( bat_E \) is calculated in terms of energy impact \( EI_E \) and a device specific constant \( \beta_E \) as given in equation (10).

\[
bat_E = \beta_E \times (EI_E)^{-1} \tag{10}
\]

Total Turnaround Time. The raw data generated by the accelerometer is partially processed by the IoT device and is then transferred to edge devices for further processing. The final processing takes place in a cloud datacentre which also saves the result for further processing (e.g. for cross-population analytics that can generate better predictive models). The total turnaround time \( T^3 \) is given by summing the time taken by IoT device \( T_D \), edge device \( T_E \) and cloud datacentre \( T_C \) from data generation to data computation and is given by equation (11).

\[
T^3 = T_D + T_E + T_C \tag{11}
\]

Each layer performs some operation and then sends the data to the upper layer. The time taken by IoT device \( T_D \) is given in equation (12). The total time is equal to the cycle length as it includes both computation time and time to send data to the edge device.

\[
T_D = cycle\_length \tag{12}
\]

The time taken by edge and cloud datacentre are given in equation (13a) and (13b) respectively.

\[
T_E = T_E(comp) + T_E\rightarrow C(trans) \tag{13a}
\]

\[
T_C = T_C(comp) \tag{13b}
\]

Where, \( T_E\rightarrow C(trans) \) is the transmission time from edge to cloud and \( T_E(comp) \) is the computation time taken by either Edge device or Cloud datacentre. For the sake of simplicity, we are not considering any waiting time or queueing time at any part of the IoT infrastructure.

Cost. Performing the operations on either IoT device, edge or cloud datacentre incurs some cost in terms of electricity charge, set up cost, cloud VM cost or storage cost. There is an additional cost associated with the data transfer. The total cost \( Cost_{Total} \) is given by the sum of the cost incurred by an IoT device \( Cost_D \), edge device \( Cost_E \), cloud datacentre \( Cost_C \) and communication cost \( Cost_{comm} \). The cost incurred by an IoT device is determined in terms of electricity cost in charging the device plus a fixed set up cost. The electricity cost depends on the power consumed (Energy Impact) by the device and the per unit electricity rate \( \rho_{elec} \). The value of \( Cost_D \) is given in equation (14a). Similarly, the cost for an edge device depends on the set up cost and electricity cost as given in equation (14b). The cost for cloud datacentre can be given in terms of launching cost and the processing and storage cost of a VM as given in equation (14c).

\[
Cost_D = (EI_D \times \rho_{elec}) + Cost_D(setup) \tag{14a}
\]

\[
Cost_E = (EI_E \times \rho_{elec}) + Cost_E(setup) \tag{14b}
\]

\[
Cost_C = Cost_C(VM_{proc}) \tag{14c}
\]

Data is transferred from the IoT device to the edge, and from the edge device to the cloud. For an IoT device, the data transfer costs drain the battery (see sustainability above) but the data transfer from edge device to cloud datacentre costs not only in terms of the edge device’s battery charge cost, but also the network charge e.g. for transferring data over a 3G or 4G network. The communication cost is calculated by multiplying the quantity of data transferred \( (msg\_count_E) \) with the data rate charge \( \rho_{data} \) as given in equation (15).

\[
Cost_{comm} = (msg\_count_E + msg\_header) \times \rho_{data} \tag{15}
\]
REFERENCES

[33] “Pay as you go rates on three,” http://www.three.co.uk/Store/Pay_As_You_Go_Price_Plans.
[34] Path2Iot-data. [Online]. Available: https://ln.sync.com/dl/278ca25a0/4hrnp9g-myx3155n-qwa9w69-mx3qe3h4s

Devki Nandan Jha is a PhD student in the School of Computing Science at Newcastle University, Newcastle Upon Tyne, UK. He received his M.Tech degree in Computer Science and Technology from Jawaharlal Nehru University, New Delhi, India. His research interests include cloud computing, big data analytics, machine learning and Internet of Things.

Peter Michalák is a PhD student at the EPSRC Centre for Doctoral Training in Cloud Computing for Big Data at School of Computing Science, Newcastle University. He previously worked as an R&D software developer in Tieto Finland Oy, received B. Eng. in Computer Software Engineering from JAMK University of Applied Sciences, Finland, and Bc. (with distinction) in Computer Engineering from the University of Zilina, Slovakia. Peter’s research interests include distributed computing, Internet of Things, and real-time event processing.

Zhenyu Wen is currently a postdoc researcher with the School of Computing, Newcastle University, UK. He received M.S and Ph.D. degrees in computer science from Newcastle University, Newcastle Upon Tyne, UK in 2011 and 2015 respectively. His current research interests include Multi-object optimisation, Crowdsourcing, AI and Cloud computing.

Rajiv Ranjan is a Full professor in Computing Science at Newcastle University, United Kingdom. Before moving to Newcastle University, he was Julius Fellow (2013-2015), Senior Research Scientist and Project Leader in the Digital Productivity and Services Flagship of Commonwealth Scientific and Industrial Research Organisation (CSIRO) and Australian Government’s Premier Research Agency. Prior to that he was a Senior Research Associate (Lecturer level B) in the School of Computer Science and Engineering, University of New South Wales (UNSW). Dr. Ranjan has a PhD (2009) from the department of Computer Science and Software Engineering, the University of Melbourne.
Paul Watson is Professor of Computer Science, Director of the Informatics Research Institute, and Director of the North East Regional e-Science Centre. He also directs the UKRC Digital Economy Hub on “Inclusion through the Digital Economy”. Prior to moving to Newcastle University, Paul worked at ICL as a system designer of the Goldrush MegaServer parallel database server. He previously gained a Ph.D. and became a Lecturer at Manchester University. His research interests are in scalable information management including data-intensive e-Science, dynamic service deployment and e-Science applications. In total, Paul Watson has over forty refereed publications, and three patents.