Edge Process Management: A Case Study on Adaptive Task Scheduling in Mobile IoT

Jakob Mass, Chii Chang and Satish Srirama

Institute of Computer Science, University of Tartu

December 5, 2018
Outline

- Background:
  - Service-oriented Internet of Things
  - Edge Process Management
  - Research Question
- Mobility-aware Scheduling Scheme
- System Architecture Overview
- Experiments, Results
- Conclusion
Service-oriented IoT

- Implement device functions as Web Services
  - e.g. RESTful HTTP API
  - Some Standard: W3C Web Of Things, FiWare, ...
- Discovery, Query
- Composition
Business Process Management Systems For IoT (BPMS4IoT)

- Compose IoT device functions into a sequence of tasks and decisions - a workflow
- Standards such as BPMN 2.0 for defining the workflow (process)
- Software that orchestrates, manages, executes BPMN processes are known as Business Process Management Systems
- BPMS are currently the most widely used approach for workflow management in IoT
- Traditionally hosted in remote centralised server
Things in Processes

Things may participate in a process either by:

- Hosting atomic services (e.g. temperature sensor, door actuator)
  - Clients invoke the services as part of their process (e.g. a cloud-hosted process engine, user's phone)
  - Common if hardware capabilities are limited
- Running the process on the device:
  - By directly interpreting & executing a process model (BPMN 2.0) using a Process Engine
  - By running executable code corresponding to a standard-based process model

Edge Process Management (EPM)

EPM - processes are managed at the edge network of IoT

- Distribute tasks, subparts of process to other devices in edge network
- Reduce server-side bandwidth
- Reduce client latency

Related domains, use cases

- Internet of Medical Things, Smart traffic control, Disaster Recovery
- **Fog Computing**: a node distributes computational task to the proximal fog server
Edge Process Management (EPM) Challenges

However, the edge network environment is highly dynamic, so the process management needs to handle context factors such as mobility and interrupted connections.

When a mobile node executes a task involving nearby fog servers, the result is affected by:

- Fog server hardware configuration
- Fog server workload
- Signal strength
- Size of the computing task
- Movement trajectory
Mobility-related challenges

Executing tasks while signal area is encountered briefly:
- task fails
- task re-executed locally or at next fog server
- resources wasted

Execution with weak signal:
- poor performance
- delays
Mobility-related challenges

Executing tasks while signal area is encountered briefly:

- task fails
- task re-executed locally or at next fog server
- resources wasted

Execution with weak signal:

- poor performance
- delays

Research Question

How do we improve the cost-performance efficiency of task execution in EPM, considering run-time context factors and without continuously relying on the distant cloud?
To address the RQ

- Propose a fog server selection scheme based on a multi-criteria decision model.
- Provide an Mobile Fog system architecture using the scheme
- Evaluate the system and scheduling scheme in a simulated environment
Fog Server Selection

Given a set of known fog servers and data about user mobility, decide which server to invoke for its distributed task.

Runs continuously, has 2 phases:

1. Candidate Fog Server set formation
2. Fine-grained, proximal decision-making
Fog Server Selection - Phase one

Given:

- Our movement trajectory
- Set of fog servers $S$ in the region

For each $s \in S$, we can estimate following numeric values:

- $E(s)$ - Energy (battery) value when entering coverage of $s$
- $M(s)$ - Meeting Duration (time spent in coverage)
- $D(s)$ - Mean distance to $s$ during movement in coverage area
- $B(s)$ - Maximum bandwidth capability of $s$
Phase 1 - Cont-d

Based on these parameters, we define a normalized parameters set $\mathcal{A}$:

$$\mathcal{A}_s = \{e_s, m_s, d_s, b_s\}$$

$e_s, m_s, d_s, b_s$ are the parameters from previous slide, but scaled to $[0, 1]$

Now, for each fog server, $s \in S$ we define the score $\beta_s$:

$$\beta_s := \sum_{a \in \mathcal{A}_s} a \times \omega,$$

Where $\omega$ is a vector $\omega \in \mathbb{R}^4$, that assigns a weight to each parameter in $\mathcal{A}$. $\omega$ allows fine-tuning the model to emphasize a certain factor, such as battery life, for example.
Initial candidate set

- Using the $\beta_s$ score, we now want to choose the servers with higher scores.
- So, we can specify threshold $\gamma$, that represents the QoS requirement of our application, and choose a set of candidate servers that are above this threshold:
  \[ S' = \{ s \mid \beta_s \geq \gamma \}, \quad S' \subseteq S. \]
- This resulting candidate set concludes phase one of the selection scheme.
Phase two

Initiated whenever we enter the coverage area of a candidate server in $S'$. The Mobile Thing fetches the current status of the fog server, introducing new parameters:

- $Q(s)$ work queue size of $s$ - how much time for already queued tasks on the server to finish
- $J(s)$ task completion time - how much time $s$ will take for processing our task (incl. communication)

With this additional info, we want to update the $\beta_s$ score so that overloaded servers are avoided.
Phase two (2)

Let’s define a 2nd weight to adjust the score:

\[
w(s) = \begin{cases} 
1 + \lambda(1 - \frac{Q(s)+J(s)}{M(s)}) & \text{iff } s \text{ status information is available,} \\
1 & \text{otherwise}
\end{cases}
\]

\(\lambda\) amplifies the penalty that waiting in a task queue has.

- \(w \geq 1\), if processing our task takes less than the meeting duration
- \(w \leq 1\), if processing our tasks takes longer than the time we are in range (can also be negative)

With the new weight, we find the final score:

\[
\beta_s^* := w(s)\beta_s, 
\]

(2)
Phase two decision-making

- We could choose the server $s$ with $\max \beta_s^*$.  
- Alternatively, set 2nd threshold $\gamma^*$ and filter another candidate set:  
  $S'' = \{ s \mid \beta_s^* \geq \gamma^* \}$
- select an element of $S''$ based on a parameter of interest, e.g. the geographically closest one
- If $S'' = \emptyset$, we can either deem the task as failed or to process it locally.
Architecture
Experiments

To see the impact of the proposed adaptive scheduling, we conducted experiments in a simulation tool - **ONE Simulator** [2].

- Discrete Event Simulator for Delay-Tolerant Protocols
- mobility models (incl map-based)
- communication interfaces, connections, messaging
- Energy model (based on communication)

However, ONE has no model-based process execution or process-level messaging!
ONE BPM

We extended ONE by implementing the Workflow Manager Component for each host.

- Based on Flowable Java engine
  - https://www.flowable.org/
  - BPMN 2.0 model execution
- Map BPMN message tasks to ONE simulator messages
- Simulated BPMN processing tasks - set the task size

The other components of the architecture were also implemented for the ONE hosts (e.g. proximal discovery and advertising)
Scenarios: Overview

We examined two types of scenarios:

- isolated cases described earlier (1 mobile thing, 2 fog servers)
- larger number of mobile hosts, servers and grid map

Compared our scheme against a baseline naïve approach

- always tries to distribute the task to the first fog server it encounters
- actually, this is how our scheme also behaves if $\gamma = 0.0$

- Our adaptive model is set to use $\gamma = 2.5$
Experiment Process Model
Case 1: Handling brief contact times

S0 signal reception is poor, s1 is located exactly on trajectory
Case 2: Handling poor signal

S0 is located nearer, so task can finish, but s1 is still closer to path
Case 3: Overloaded servers

Both s0 and s1 are equally close to trajectory, but s0 has other tasks in queue
Case 4: Multiple Mobile Hosts

- 50 mobile hosts, 9 fog servers
- Manhattan-style movement map
- 4 hrs simulation clock time (single host can start 16 processes)
Case 4: Process Success Rate

Figure: Average distributed task completion for Mobile Thing Hosts

Baseline achieves $\leq 50\%$, adaptive approach reaches $75\%$ at $QoS = 2.5$
Case 4: Distribution Among Servers

**Figure:** Average distributed task completion for Mobile Thing Hosts

Baseline: high variance, adaptive approach balances load better.
Case 4: Energy consumption

Figure: Energy Consumption for MoTHs

QoS values directly influence the energy consumption. Baseline outperformed even with larger messages.
Conclusion

- The proposed scheme showed benefits in terms of task execution time, energy consumption, but also load balancing.
- Improvement is especially significant if decisions leading to dropped connections can be avoided.
- However, choosing the appropriate thresholds, $\gamma$ values affect performance, may be dependent on use case or user preference.
- E.g. very high QoS values may be desired if battery life is critical.
- This scheme suits scenarios where some delay-tolerance is acceptable.
Future Work

- Improved decision-model, e.g. more detailed task and communication modelling
- V2V use cases, where both server and client are mobile
- Mobility prediction models
Thank you for listening!