QoS-Aware Resource Allocation for Mobile IoT Pub/Sub Systems

Raphael Gomes, Georgios Bouloukakis, Fabio Costa, Nikolaos Georgantas, Ricardo Da Rocha

QoS-Aware Resource Allocation for Mobile IoT Pub/Sub Systems

Raphael Gomes[1], Georgios Bouloukakis[2,3], Fábio Costa[4], Nikolaos Georgantas[2], and Ricardo da Rocha[5]

[1] Instituto Federal de Goiás - Câmpus Goiânia, Brazil
raphael.gomes@ifg.edu.br
[2] Inria Paris, France
firstname.lastname@inria.fr
[3] Donald Bren School of Information and Computer Sciences, University of California, Irvine, USA
gboulouk@ics.uci.edu
[4] Instituto de Informática, Universidade Federal de Goiás, Brazil
fmc@inf.ufg.br
[5] Instituto de Biotecnologia, Universidade Federal de Goiás, Catalão, Brazil
rcarocha@acm.org

Abstract. IoT applications are usually characterized by large-scale demand and the widespread use of mobile devices. Similarly, performing interaction among application and system components in a decoupled and elastic way, and enforcing Quality of Service (QoS) usually also become issues. Hence, paradigms such as pub/sub on top of cloud resources represent a suitable strategy for application development. However, management of QoS-aware resource allocation for pub/sub systems remains challenging, especially when system peers connect in an intermittent way. In this paper, we propose a new approach for resource allocation focusing on end-to-end performance in face of peers' disconnections. We evaluate and demonstrate the benefits of our approach using simulations. QoS enforcement was achieved in almost all scenarios, and we have shown that our approach can help reasoning about efficient resource allocation.

Keywords: Publish/Subscribe Middleware, Resource Allocation, Mobile Connectivity, Quality of Service

1 Introduction

The widespread use of smart mobile devices paved the way for the development of smartphone applications that can be accessed anywhere anytime. Additionally, such applications access embedded sensors/actuators for providing environmental-related information, opening new business and market opportunities in the so-called *Internet of Things* (IoT) [4]. However, mobile devices may connect and disconnect intermittently for energy saving purposes, and may be forcefully disconnected due to connectivity issues in underlying wireless network. Accordingly, asynchronous messaging patterns, such as the *Publish/Subscribe*

(pub/sub) paradigm, enable event-buffering during disconnected periods and event-delivery during connected periods. In particular, pub/sub provides loosely coupled interaction in both time and space among publishing data sources and subscribing data sinks [15]. As a result, several industry standards have adopted pub/sub as part of their interfaces.

Existing middleware protocols such as MQTT [6] and AMQP [30], as well as tools and technologies such as RabbitMQ [27], Kafka [3] and JMS [26] follow the pub/sub paradigm. To support the development of effective applications under the constraints found in the IoT (i.e., intermittent connectivity, obsolete data feeds, etc), pub/sub protocols provide several *Quality of Service* (QoS) features [10]. These features aim to enable application developers to tune an application by configuring its QoS metrics at different levels such as end-to-end response times and delivery success rates.

Despite the fact that the QoS features of pub/sub may enable timely data delivery, this can be affected (as well as other QoS metrics) by the the hardware resources used to deploy software components such as message brokers [19]. Besides, although mobile devices have increasing processing, storage, and communication capabilities, they are not able to handle all sorts of tasks in a proper way [8], making imperative the use of external resources. The use of *Cloud computing* is the most promising solution to enable the deployment of scalable and reliable software components. This further enhances the loosely coupled interaction between publishers and subscribers, also improving reliability as deployed message brokers in the cloud are always connected to receive/forward events.

Different solutions have been proposed for the problem of resource allocation under specific QoS constraints in order to support the development of message brokers in the Cloud [16,25,28]. However, these solutions mainly focus on the satisfaction of local QoS requirements (e.g., employing load balancing strategies for individual jobs) lacking support for end-to-end non-functional properties. Proposed strategies for end-to-end QoS enforcement in pub/sub systems [7,12,19] have limited scope (e.g., changing on transport protocols) or present limitations such as incompatibility with existing middleware protocols. Additionally, in pub/sub systems, the subscribers' intermittent connectivity affects the existing resource allocation algorithms. For instance, message brokers with local subscribers that disconnect for long periods, demand additional processing and buffer resources. Thus, the design of QoS-aware resource allocation algorithm for mobile IoT applications remains an important challenge as it is necessary to satisfy elastic demands while keeping costs under control.

To deal with the above limitations, in this paper we present a QoS-aware approach for the allocation of resources for IoT applications. We consider IoT applications that run over a pub/sub system with mobile publishers/subscribers, along with message brokers that are deployed in the cloud. We formalize the resource allocation problem and estimate response times by relying on *Queueing Network Models* [20]. Our approach provides system designers with resource allocation estimates at design time enabling the achievement of accurate end-to-end runtime behavior. We evaluate and demonstrate the benefits of our approach

using simulations based on both probability distributions and parameters derived from real-world workload traces. The core contributions of the paper are:

1. Formalization of the resource allocation problem in mobile pub/sub systems.
2. A model for the response time from publishers to subscribers, which is obtained by connecting queuing models that represent cloud resources and peers' disconnections.
3. A cost-effective resource allocation strategy based on the splitting of end-to-end QoS thresholds and on resource selection in a multi-cloud environment.
4. A simulation environment for the evaluation of QoS properties in pub/sub systems with awareness of the connectivity status of communicating peers.

The rest of this paper is organized as follows: Section 2 introduces the pub/sub system model and the problem of allocating resources for message brokers. In Section 3, a formalization of this problem is presented. Our resource allocation approach is provided in Section 4, which takes into account the peers' connectivity. Finally, the validation and evaluation of our approach is considered in Section 5, followed by conclusions and future work in Section 6.

2 System Model

Mobile IoT applications are typically deployed on resource-constrained devices with intermittent network connectivity. To support the deployment of such applications, the pub/sub interaction paradigm is often employed, as it decouples mobile peers in both time and space. In a pub/sub system, multiple peers interact via an intermediate *broker* entity – publishers produce *events* characterized by a specific *filter* to the broker. Subscribers subscribe their interest for specific filters to the broker, who maintains an up-to-date list of subscriptions.

To support distributed applications spanning a wide-area, the pub/sub system has to be implemented as a set of independent, communicating brokers, forming the *broker overlay*. As depicted in Fig. 1, in such architectures [5,9], peers can access the system through any broker that becomes their home broker. Then, based on the peers' input the pub/sub system performs several processes: *i)* subscriptions are spread to a subset of existing brokers through a *subscription partitioning* process; *ii)* published events correspond to subsets of subscribers determined through a *matching* process; *iii)* the produced events are delivered to all the matched subscribers through an *event routing* process. Regardless of the event routing algorithm in use, produced events pass through a specific path of brokers towards the subscribers. For example, in Fig. 1 the produced events pass through brokers b_2, b_5 and, finally, b_7, which is s_1's home broker.

Building an IoT application over a pub/sub infrastructure, requires the selection of an appropriate protocol (e.g., MQTT). Such a protocol enables peers to access the broker overlay and push/receive events. Additionally, to create the pub/sub broker overlay, it is essential to select the corresponding *message broker* implementation (e.g., ActiveMQ, VerneMQ, HiveMQ, etc). Every message broker has different capabilities, such as: compatibility with different protocols, support for clustering (i.e., forming a broker network), provision of performance

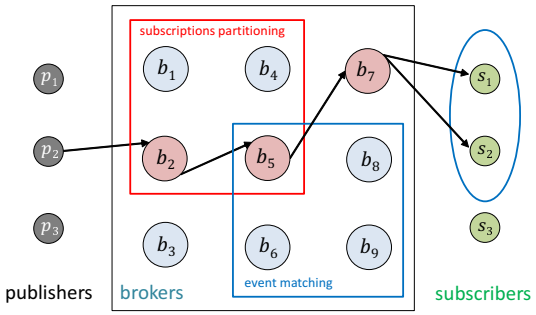

Fig. 1. Pub/Sub System.

features, etc. Finally, an important design-time decision is the deployment of each broker to an appropriate machine. A common tactic of pub/sub developers is to deploy or assume the deployment of brokers in the cloud.

To perform cloud resource allocation, different technologies (e.g., Virtual Machines - VMs [28] and containers [25]) can be used, each with its strengths and weaknesses. In this paper, we use VMs as the unit of allocation, although we keep the solution as generic as possible to admit other alternatives. Terms such as *resource* and *VM* are used interchangeably in the text. For simplicity, we assume that each resource is used by a single broker via on-demand resource instantiation. Accordingly, the pricing of resource allocation is estimated taking the one-hour utilization of an instance of the given resource type.

In IoT scenarios, system deployment usually aims to achieve co-location of *things* and allocated resources. This strategy is used to decrease the communication demand on devices with hardware and power limitations. Another motivation is that whilst data-processing speeds have increased rapidly, the bandwidth to carry data to and from datacenters has not increased at the same speed [29]. For these reasons, in our approach resource allocation is limited to resources found inside a single region. For cloud vendors such as Amazon [2], which splits a region into multiple availability zones, we adopt a single availability zone. Hence, we neglect data propagation delays inside pub/sub system when evaluating QoS.

Based on the above, several questions arise: *i)* what is the amount of resources that a developer should allocate for an IoT application that runs over a pub/sub system? *ii)* Can we ensure specific end-to-end response times between publishers and subscribers by relying on the resulting resource allocation? *iii)* What is the cost to achieve such end-to-end QoS? *iv)* Does the intermittent connectivity of peers affect the cost of resource allocation?

In the next section, we delineate our approach using a formal notation for the problem with respect to the questions raised above.

3 QoS-aware Resource Allocation for Pub/Sub Systems

The lack of a direct producer/consumer relationship in pub/sub interactions makes the definition and enforcement of any end-to-end QoS policy very hard [14]. Additionally, the peers' intermittent connectivity makes the design of a fully

decoupled QoS-driven pub/sub system that can scale to different dimensions without under- or over-provisioning of resources even more challenging.

Resource allocation is carried out by translating QoS constraints into infrastructure level parameters, which are then mapped to a number of predefined resource types, described in terms of hardware capacity, cost, and location. We call the joint execution of these tasks ***resource synthesis***. It is not a straightforward task due to the large number of resource type options and intersections among them. Yet, even considering a single region of any given cloud provider, a possibly large number of resource types may need to be inspected and the best cloud vendor must be selected. To elucidate the main elements of the resource synthesis problem and precisely outline its scope, we first formalize it.

3.1 Problem Formalization

Given a pub/sub system with intermittently disconnected peers, the ***Resource Allocation Problem*** **(RAP)** consists in selecting the proper resource type and the number of resource instances, in order to deploy a network of brokers aiming at QoS enforcement and cost savings. To mathematically represent event streams, we use a topic-based subscription model, since it is efficient and simple in terms of event classification. Nevertheless, our approach can be used for any model where several classification techniques are applied. We model the parameters concerning the RAP as follows:

$R = \{r_j : j \in [1..|R|]\}$ is the set of event topics that correspond to IoT sensor data, device commands, etc.

$S = \{s_i : i \in [1..|S|]\}$ is the set of subscribers: devices or persons interested in event notifications. They subscribe to any number of event topics in R. We denote the set of topics that s_i subscribes as $R_{s_i} \subseteq R$. Let $\lambda^{sub}_{s_i,r_j}$ be the delivery rates of events to each subscriber s_i for publications to topic $r_j \in R_{s_i}$.

$P = \{p_i : i \in [1..|P|]\}$ is the set of publishers: devices/persons that publish events on some set of topics. We denote the set of topics that p_i publishes as $R_{p_i} \subseteq R$. Let $\lambda^{pub}_{p_i,r_j}$ be the publication rate of events published to to topic r_j by publisher $p_i \in P$. We assume that each p_i publishes events according to a Poisson process at each topic r_j.

We model the connectivity of pub/sub peers as follows: let ON and OFF be the states where the peer is connected and disconnected, respectively. A given peer, is connected (ON state) for an exponentially distributed time period with parameter θ^{ON} ($\theta^{\text{ON}} = 1/T^{\text{ON}}$). Upon the expiration of this time, the peer disconnects (OFF state) and stops sending or receiving relevant events for an exponentially distributed time period with parameter θ^{OFF} ($\theta^{\text{OFF}} = 1/T^{\text{OFF}}$). Accordingly, let $T^{\text{ON}}_{p_i}$, $T^{\text{OFF}}_{p_i}$ be the connection/disconnection average periods for p_i and $T^{\text{ON}}_{s_i}$, $T^{\text{OFF}}_{s_i}$ for s_i.

$B = \{b_k : k \in [1..|B|]\}$ is the set of pub/sub brokers. A broker forwards events from publishers to interested subscribers or to other brokers in the *broker network* for eventual consumption by a subscriber. We assume that each publisher/subscriber connects with a single broker that we refer to as its *home*

broker: b_{p_i} is the broker that publisher p_i publishes to and b_{s_i} is the broker that subscriber s_i receives events from. Furthermore, we define the set of publishers and subscribers connected with b_k as $P_{b_k} = \{p_i \in P : b_k = b_{p_i}\}$ and $S_{b_k} = \{s_i \in S : b_k = b_{s_i}\}$ respectively.

$V = \{v_j : j \in [1..|V|]\}$ is the set of resource types (e.g., VMs).

$\zeta[]_{v_j}$ is a vector used to describe a resource type. It contains information about hardware capacity (e.g., CPU cores and clock speed, bandwidth, etc.).

c_{v_j} is the allocation cost of an instance created using the resource type v_j.

Δ_{s_i} is the end-to-end *response time* of events matching subscriber's topics R_{s_i} from the moment they are published until s_i receives them.

Δ_{p_i,s_i} is the end-to-end response time of events matching the common subscriber's and publisher's topics (i.e., $R_{p_i} \cap R_{s_i}$) from the moment they are published by p_i until s_i receives them.

The above delays include event processing times and network delays inside the broker network. Note that in this paper we do not model processing delays of publishers/subscribers. Our main purpose is to estimate delays inside the broker network in order to allocate their resources accordingly.

$\Delta^{p_i,s_i}_{b_k,v_j}$ is the response time contribution from broker b_k on the end-to-end path between p_i and s_i. This contribution depends on the resource v_j selected for broker deployment.

Δ^{thr} is the response time threshold inside the pub/sub system for any end-to-end path between p_i and s_i. When setting this parameter, resource allocation should be performed in such a way that the end-to-end response time (Δ_{p_i,s_i}) of all events between any p_i and s_i does not exceed this limit.

x_i is an integer variable that indicates if Δ^{thr} has been satisfied for a given end-to-end path, as follows:

$$x_i = \begin{cases} 1, & \text{if } \Delta_{s_i} \leq \Delta^{thr}; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

$F = \{f_k : k \in [1..|F|]\}$ is the set of mappings (result of RAP), where each f_k is a pair (b_k, v_j) representing the mapping of a broker to an allocated resource type (which includes the number of deployed resource instances).

φ_{v_j} is an objective function depicting the profit in allocating resource v_j. It is used to distinguish the best option among feasible solutions.

$\varphi(F)$ is the composed profit of all resources in F. In this way, the optimal solution satisfies the QoS threshold for all subscribers and gives the best $\varphi(F)$.

RAP can be more formally expressed as an optimization problem as follows:

$$\begin{aligned}
\text{Maximize} \quad & \varphi(F) = \sum_{b_k \in B} \sum_{v_j \in V} \varphi_{v_j} y_{kj}; \\
\text{subject to} \quad & \sum_{s_i \in S} x_i = |S|; \\
& \sum_{v_j \in V} y_{kj} = 1, && k \in [1..|B|]; \\
& y_{kj} \in \{0,1\}, && k \in [1..|B|], \\
& && j \in [1..|V|].
\end{aligned}$$

The variable y_{kj} is either 1, implying b_k is mapped to v_j, or 0 otherwise.

As RAP is clearly an optimization problem, we argue that it is not possible find an optimal solution for an instance of this problem in polynomial time. Nevertheless, in the next section, we provide a feasible approach based on a reduction to a version of the knapsack problem and on an analytical model for QoS estimation for solving this problem.

4 Resource Synthesis with Intermittent Connectivity

Our approach focuses on the QoS-aware allocation of resources for deploying message brokers at design time. Initially, a system designer provides the topology and characteristics of the pub/sub system (brokers, peers' intermittent connectivity, etc), its routing algorithm [5] (e.g., rendezvous nodes) and the end-to-end response time threshold (Δ^{thr}). Then, our approach provides a specification of the corresponding resources, in terms of types and quantity. After deploying brokers based on the specification, end-to-end response times must be below the specified Δ^{thr} at runtime. In this paper, we are agnostic with respect to the algorithm used to route events. Hence, we assume that the end-to-end paths between publishers and subscribers (i.e., the broker paths) are estimated somehow using the provided routing algorithm. Additionally, we acknowledge that the resulting resource allocation may be invalidated if different routing paths are used at runtime. However, we argue that complementary solutions may benefit from cloud elasticity to perform adaptation of the resource allocation.

To provide an efficient solution for the above-described problem without the costs associated with an exhaustive search strategy, we rely on well-known solutions to the Multiple-choice Multi-dimension Knapsack Problem (MMKP) [24]. MMKP is a NP-Complete problem defined as follows: given a set H of items divided in h categories Q_q, where each category $Q_q : q \in [1..h]$, has $\varkappa_q = |Q_q|$ items such as $\forall\ 1 \leq i, j \leq h$ and $i \neq j, Q_i \cap Q_j = \emptyset$ and $\cup_{i=1}^{h} Q_i = H$. Each item $o \in [1..\varkappa_q]$, from category Q_q has a non-negative profit ϱ_{qo}, and has dimensions given by a weight vector $W_{qo} = \{w_{qo}^a : a \in [1..\ell]\}$, where each element w_{qo}^a is also a non-negative value. The knapsack dimensions are given by the vector $A = \{A^a : a \in [1..\ell]\}$. The goal in MMKP is to pick exactly one item from each category in order to maximize the total profit, subject to knapsack dimensions.

In the provided approach, we construct an MMKP instance from a RAP instance. To do so, we rely on results related to solving MMKP. We use the WS-HEU heuristic [31] to efficiently select the resource types in order to reach the thresholds for each constraint. We chose this heuristic because it solves the MMKP problem with an accuracy of 96% compared to the optimal value, and can provide worst-case complexity of $O(|S| \times |B|^2 \times (|V|-1)^2)$. The construction of an MMKP instance from a RAP instance is carried out as follows. Let n be the number of categories (h). We map each $b_k \in B$ as a category Q_q with same number of items (generated from resource types, i.e., each $v_j \in V$ is mapped as a different item o for each category). In this way, we remove the property of empty set in the items intersection of categories, but this change does not affect the

problem since constraints are evaluated separately for each category. We generate the weight vector W_{qo} for each item by using the average delays of each routing path in which the corresponding broker participates, so that $[\Delta^{p_1,s_1}_{b_k,v_j}..\Delta^{p_{|S|},s_{|S|}}_{b_k,v_j}]$ is mapped to $[w^1_{qo}..w^\ell_{qo}]$, and φ_{v_j} is mapped to ϱ_{qo}. Finally, the response time threshold is multiply mapped as knapsack dimensions $A = \{A^a : a \in [1..\ell]\}$.

End-to-end QoS evaluation is carried out in our approach by separately addressing each possible routing path in the pub/sub system. Although this may sound inefficient, we argue that by using the chosen heuristic we can reach the resource synthesis result in admissible time. Additionally, this task is performed at design time, when low overhead is not a major concern. We discuss this further in Section 5 when presenting the experimental evaluation. By using the WS-HEU heuristic, an initial solution is pursued using a greedy strategy based on the ratio between resource cost and QoS offering. In cases where even considering all resource types no feasible solution is found, an event of unaccomplished resource synthesis is triggered. We deal with this problem through the inclusion of new broker instances. To this end, we assume a round-robin load balancer with negligible overhead. The selection of the broker to be replicated is carried out by taking the broker with the worst QoS, which is most likely one of the edge brokers due to the effects of peers' disconnections. Additional broker instances are added until a feasible solution is found. When this happens, there is an attempt to improve the solution by performing simulated annealing [1]. The entire procedure is performed for the resource types from each cloud provider.

The resource synthesis output is a mapping of each broker (and its additional instances) to a selected resource type, taking the less costly result. By using this strategy, resource allocation is performed through the balancing of the contribution to the end-to-end constraints among all participating brokers. As can be seen, a key aspect of this approach is QoS estimation. For this purpose, we propose the performance model described next.

4.1 Pub/Sub QoS Estimation

In this subsection, we present our queueing network model which is used as input to the resource allocation algorithm. By relying on our previous work [9], we model the performance inside a pub/sub broker network by relying on queueing theory. In particular, each queue processes events through a dedicated *server*. Each server supports a specific service rate (λ) which is exponentially distributed. All queueing centers apply a first-come-first-served (FCFS) queueing policy. We model the network transmission delay between brokers ($b_k \rightarrow b_i$) using an $M/M/1$ queue and between b_{s_i} (s_i's home broker) $\rightarrow s_i$ (subscriber) using an ON/OFF queue. With regard to [9], in this paper we slightly modify our formal model and queueing models. More specifically, in order to process incoming events at a broker node b_k we use the $M/M/c$ queueing model: c is the number of servers of the queue and each server has an independently and identically distributed exponential service-time distribution with mean $1/\mu$. We correspond c to the number of VM cores in which the broker is deployed.

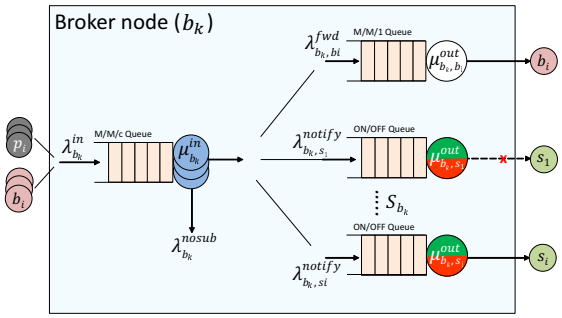

Fig. 2. Queneing Network for Pub/Sub Broker Node.

As depicted in Fig. 2, we model each broker b_k using a single inbound queue $q^{in}_{b_k}$ (M/M/c) and multiple outbound queues $q^{out}_{b_k}$ (M/M/1 and ON/OFF). Let $\lambda^{in}_{b_k}$ be the arrival rate of events at $q^{in}_{b_k}$, represented as the sum of all event publication/forwarding rates over all publishers/brokers. Forwarding, replication, or dropping of events based on current subscriptions occurs at the exit of $q^{in}_{b_k}$. Let $\mu^{in}_{b_k}$ be $q^{in}_{b_k}$'s service rate for analyzing an incoming event and determining where to forward it (e.g. based on a topic routing tree). We assume that the service rate is exponentially distributed across all topics with parameter $\mu^{in}_{b_k}$. Events that do not match b_k's subscriptions are dropped with rate $\lambda^{nosub}_{b_k}$.

Broker b_k forwards events matching subscriptions served by other brokers with rate $\lambda^{fwd}_{b_k,b_i}$ through an outbound M/M/1 queue $q^{out}_{b_k,b_i}$. These events are sent from b_k to any $b_i \in B, b_i \neq b_k$. Let $\mu^{out}_{b_k,b_i}$ be $q^{out}_{b_k,b_i}$'s service rate for transmitting an event to b_i. For each of b_k's local subscribers, $s_i \in S_{b_k}$ & $b_{s_i} = b_k$, b_k forwards events matching subscriptions to $q^{out}_{b_k,s_i}$ with rate $\lambda^{notify}_{b_k,s_i}$ for transmission to s_i. We model each $q^{out}_{b_k,s_i}$ using an ON/OFF queue for transmitting events based on s_i's intermittent connectivity (i.e., $T^{\text{ON}}_{s_i}$, $T^{\text{OFF}}_{s_i}$). Let $\mu^{out}_{b_k,s_i}$ be $q^{out}_{b_k,s_i}$'s service rate for transmitting an event to s_i. Note that $\mu^{out}_{b_k,s_i}$ and $\mu^{out}_{b_k,b_i}$ rates model the transmission delay of events inside the network.

To build the broker model of Fig. 2, we use three different types of queueing models. Based on standard solutions, each queueing type evaluates several performance metrics (e.g., response time) through analytical models. Particularly, with regard to response time in M/M/1 queues [20], the time that an event remains in the system (corresponding to queueing time + service time, also called average delay) is given by:

$$\Delta_{q_{mm1}}(\mu, \lambda) \;=\; \frac{1}{(\mu \;-\; \lambda)} \tag{2}$$

Regarding the M/M/c queue, the time that an event is remain in the system is given by [18]:

$$\Delta_{q_{mmc}}(\mu, \lambda, c) \;=\; \frac{1}{\mu} + \left(\frac{r^c}{c!(c\mu)(1-\rho)^2}\right) p_0 \tag{3}$$

where c is the number of servers, $r = \lambda/\mu$, $\rho = r/c$ and the steady-state probability p_0 can be found in [18]. Finally, regarding the ON/OFF queue, the time that an event is remain in the system is given by [9]:

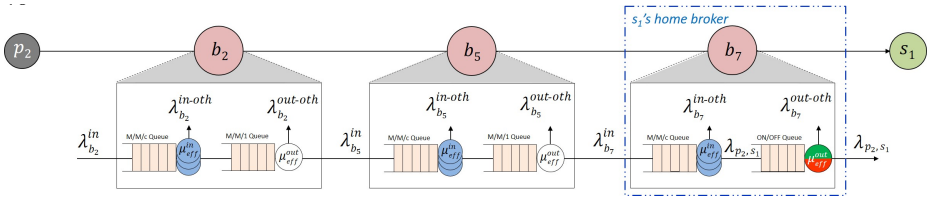

Fig. 3. Queneing network for the end-to-end interaction: p_2 to s_1.

$$\Delta_{q_{onoff}}(\mu, \lambda, T^{\text{ON}}, T^{\text{OFF}}) = \frac{\frac{T^{\text{OFF}2}}{T^{\text{ON}}+T^{\text{OFF}}} + \frac{T^{\text{ON}}+T^{\text{OFF}}}{\mu T^{\text{ON}}}}{1-\lambda\frac{T^{\text{ON}}+T^{\text{OFF}}}{\mu T^{\text{ON}}}} \tag{4}$$

In the next subsection, we show how these analytical models can be used to estimate end-to-end response times.

End-to-end Response Time. We propose a strategy to allocate resources for deploying pub/sub brokers based on the response time threshold (Δ^{thr}) applied inside the pub/sub system. Hence, in order to allocate resources it is essential to calculate the expected delay between p_i and s_i (Δ_{p_i,s_i}) inside the broker network by considering the intermittent connectivity of peers. Based on the broker's queueing model (see Fig.2), s_i's connectivity affects the resulting delay – i.e., during $T^{\text{OFF}}_{s_i}$, b_k buffers events. Furthermore, events may pass through a subset of brokers, until they arrive to s_i's home broker (b_{s_i}). Therefore, for a given s_i, the average delay inside a broker node b_k can be estimated as follows:

$$\Delta_{b_k} = \begin{cases} \Delta_{q^{in}_{b_k}} + \Delta_{q^{out}_{b_k,b_i}}, & \text{if } b_k \neq b_{s_i} \\ \Delta_{q^{in}_{b_k}} + \Delta_{q^{out}_{b_k,s_i}}, & \text{if } b_k = b_{s_i} \end{cases} \tag{5}$$

Subsequently, for a given interaction between p_i and s_i, events pass through a subset of intermediate brokers (b_i) and s_i's home broker (b_{s_i}). Let λ_{p_i,s_i} be the rate of events from p_i to s_i. To estimate Δ_{p_i,s_i}, it is essential to apply the so called *effective* service rate ($\mu^{in/out}_{b_k-eff}$) at each inbound/outbound queue of b_k. Such a service rate is calculated only for the λ_{p_i,s_i} rate but by taking into account other rates of events $\lambda^{in/out-oth}_{b_k}$ (lack of subscriptions, forwarding rates to other brokers, etc) [9].

To demonstrate our approach, we provide the following example: to Δ_{p_2,s_1}, the resulting queueing network is depicted in Fig.3. Hence, Δ_{p_2,s_1} is given by:

$$\Delta_{p_2,s_1} = \Delta_{b_2} + \Delta_{b_5} + \Delta_{b_7}$$

Based on Fig.3, b_2, b_5 are intermediate brokers and b_7 is s_1's home broker ($b_7 = b_{s_1}$). Hence, by relying on (5), we estimate the end-to-end delay using the corresponding queues. As already pointed out, $q^{in}_{b_k}$ is an M/M/c queue, $q^{out}_{b_k,b_i}$ is an M/M/1 queue and $q^{out}_{b_k,s_i}$ is an ON/OFF queue. Hence, by relying on (2), (3) and (4) the average delay of an intermediate broker, e.g., b_2 is given by:

$$\Delta_{b_2} = \Delta_{q_{mmc}}(\mu^{in}_{b_2-eff}, \lambda_{p_2,s_1}, c_{b_2}) + \Delta_{q_{mm1}}(\mu^{out}_{b_2-eff}, \lambda_{p_2,s_1})$$

And the average delay of b_{s_1} is given by:

$$\Delta_{b_{s_1}} = \Delta_{q_{mmc}}(\mu^{in}_{b_7-eff}, \lambda_{p_2,s_1}, c_{b_7}) + \Delta_{q_{onoff}}(\mu^{out}_{b_7-eff}, \lambda_{p_2,s_1}, T^{\text{ON}}_{s_1}, T^{\text{OFF}}_{s_1})$$

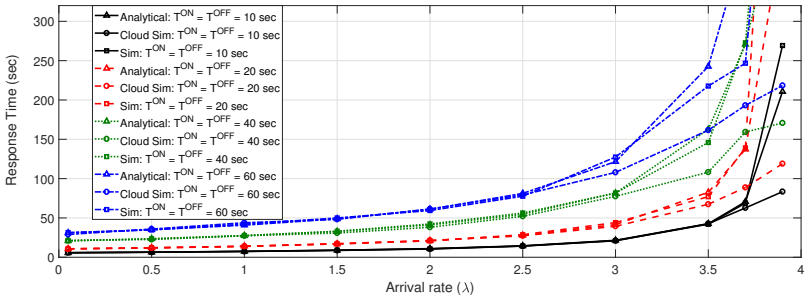

Fig. 4. Comparison of results from M-CloudSim, Analytical Model and MobileJINQS.

5 Experimental Evaluation

The goal of our experiments is to study the effects of our approach on QoS enforcement, on the reduction of the resource allocation cost, and on performance. In this section, we first introduce a simulation environment developed to enable this evaluation and then discuss the experiments and achieved results.

5.1 Simulating Intermittent Connectivity over Pub/Sub Systems

Experimental evaluation requires tools that facilitate the design of experiments while making them repeatable and precise. Simulators are useful tools to build such evaluation environments in the cloud context [32]. However, to the best of our knowledge, there is no simulation environment for cloud-based pub/sub systems enabling the modeling of the peers' connectivity. For this reason, we developed our own simulator, called *Mobile CloudSim* (M-CloudSim)[6], as an extension of the CloudSim simulation toolkit [11] with the following features.

1) Application model: Simulations in CloudSim are implemented by modeling a set of cloudlets (application tasks) and VMs. On top of it, a cloud mediator is defined to schedule cloudlets according to the available VMs in a discrete-event simulation dynamics. Although this model is feasible to represent the main aspects of cloud applications, it does not allow indirect communication modeling as well as routing in message brokers. To solve this limitation, in our extension we have distinguished two cloudlet categories: event processing and event transmission. We explicitly represent the events routing paths, creating and submitting cloudlets from both categories to simulate event forwarding.

2) Modeling intermittent peer connectivity: one of the main goals in our evaluation is to evaluate the effects of peer disconnection. We handle this by simulating changes on peer connection state. On the simulation, peer connection is checked before the processing, transmission, and delivery of events on home brokers. We only simulate changes of connectivity states for subscribers since during the publishers' disconnections there is no event-transmission to brokers and, therefore, no relation with resource allocation.

We kept the M-CloudSim functionalities limited to the scope of this paper. As a way of demonstrating the validity of the results provided, we have performed

[6] `https://github.com/raphaeldeaquino/mcloudsim`

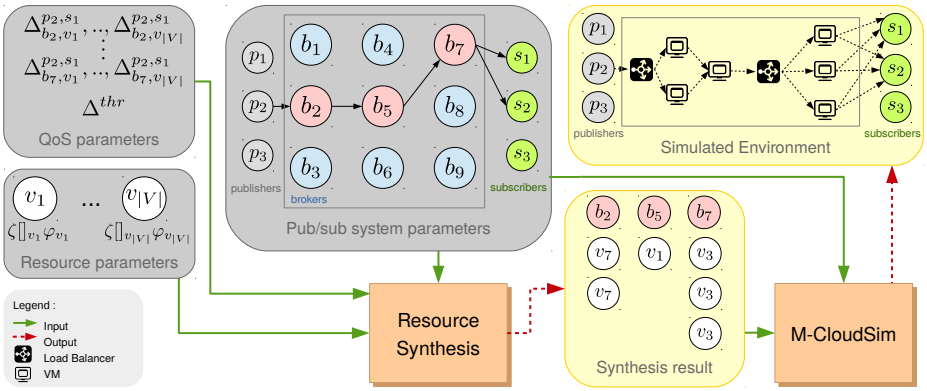

Fig. 5. Resource synthesis validation methodology.

an experimental comparison with two other tools: using the analytical model presented in this paper and using MobileJINQS [9], an open-source library for building simulations encompassing the constraints of mobile systems (but lacking resource simulation). In this evaluation, we have computed the response time in a specific path of a pub/sub system. In the simulated environment the subscriber remains in the ON and OFF states for exponentially distributed time periods $T^{\text{ON}} = T^{\text{OFF}} = 10/20/40/60$ sec. For the analytical model and the MobileJINQS simulation, processing and transmission of events are served with a mean service demand of 0.0625 sec and 0.125 sec, respectively. Accordingly, in the M-CloudSim simulation we set CPU and bandwidth characteristics in the simulated VMs to provide the same timing. We assume sufficient buffer capacities so that no events are dropped. Also events are generated by the publisher with a mean rate varying from 0.05 to 4 events per sec. By applying λ rates greater than 4 events/sec, the system saturates both in the analytical model and in the MobileJINQS simulation. Differently, M-CloudSim can simulate any number of input rates since it works for a predefined time slice by using waiting queues with "infinite" size. The mean response time of 100 independent executions (for each scenario) is depicted in Fig. 4. Confidence intervals of the simulation results are found to be very small and are not presented in the figure.

By comparing the curves for the three strategies response times, we notice that results match with high accuracy for arrival rates below to 3.5 events/sec. Differences are noticed for rates equal to or higher than 3.5 events/sec. This is acceptable since the system is close to saturation at these rates.

5.2 Resource Synthesis Validation

In this experiment, we check whether QoS enforcement is achieved as a result of the resource synthesis approach. We also evaluate the resulting profit by using the resource allocation cost as a parameter. On top of it, we check whether the peer's intermittent connectivity affects the resource allocation cost. For this purpose, we carry out the resource synthesis in different scenarios and use M-CloudSim to examine the achieved response times. We compare the following scenarios: *i)* with always-connected pub/sub peers; and *ii)* with intermittently

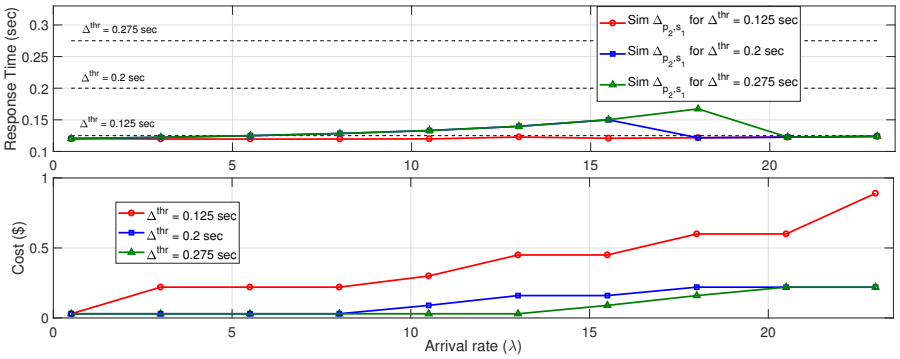

Fig. 6. Results of synthesis validation for always connected subscriber.

connected peers due to network issues. In both cases, we apply the methodology illustrated in Fig. 5: we first perform resource synthesis by using as input an end-to-end response time threshold (Δ^{thr}), the pub/sub system topology (i.e., publishers, subscribers), the flow of events, the end-to-end routing path of events, the peers intermittent connectivity and the available resource types.

For all scenarios, we applied as input the VM types provided by the market-leading vendors according to Gartner's latest report on cloud Infrastructure as a Service [21] – Amazon [2], Microsoft Azure [23], and Google Cloud Platform [17]. In this experiment, we use the available VM configurations in the São Paulo, Brazil, region on Jan-2018, with a total of 94 resource types. For the pub/sub system we use the same topology of Fig. 1 by taking only events on path $p_2 \rightarrow s_1$. We isolate other flows of this path by estimating the corresponding effective service rate (μ_{eff}) for $p_2 \rightarrow s_1$. Larger scale scenarios (with more paths) are used to evaluate the scalability of the approach in the next section. We assume that the broker network is reliable, with no broker overload. In addition, all events have the same size, which is equal to 200 bytes (approximately 200 characters of application payload). Also event processing requires 33.2 CPU instructions (average number of instructions in the method body of general Java programs [13]). We vary the arrival rate of events from 0.5 to 23 events per second. We set Δ^{thr} with an initial value of 0.025 sec. Then, we increase Δ^{thr} with increments of 0.025 sec. We perform evaluations until the synthesis result is the same for at least 5 consecutive arrival rate variations (half of the analyzed scenarios). We present the results for the two scenarios next.

Always Connected

As already pointed out, in this scenario we assume that a subscriber is always connected ($T_{s_i}^{\text{OFF}} = 0$). We use this case to analyze what is the minimum achieved response time in the given scenario and to generate a comparison baseline for the next case. Fig. 6 presents the results of simulated response times and resource allocation cost for selected thresholds. Although we have analyzed several thresholds, for better readability we present only three cases: *i)* the minimum achieved response time ($\Delta^{thr} = 0.125$); *ii)* the upper threshold analyzed ($\Delta^{thr} = 0.275$), i.e., when synthesis result is stable for at least half of the arrival

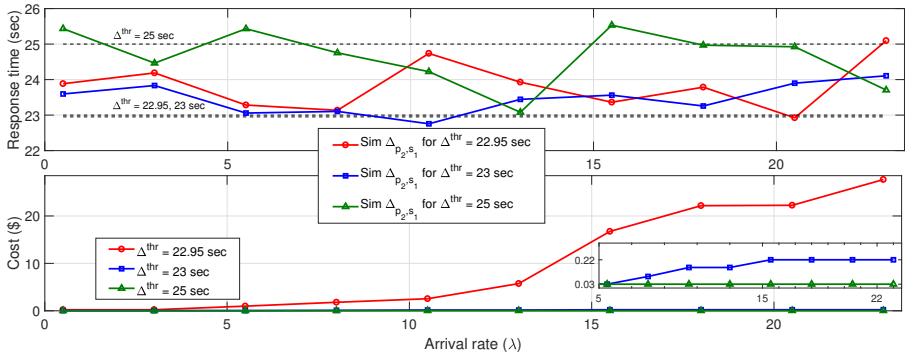

Fig. 7. Results of synthesis validation for subscriber's intermittent connectivity.

rates; and *iii)* an intermediary value between them ($\Delta^{thr} = 0.2$). We discuss only these thresholds because it is not possible to achieve smaller values with the available resource types (even with multiple instances). On the other hand, higher thresholds do not affect the synthesis result when using the considered arrival rates. As the main result, QoS enforcement was achieved in all scenarios. It is worth noting that the simulated response times are, on average, 30% better than the set of thresholds, while the resource allocation cost was, on average, \$0.40, \$0.12, and \$0.09, respectively for each threshold. As expected, the allocation cost increases for higher arrival rates. As a remark, the resource types on Amazon provided the best results in all scenarios for the São Paulo region.

Network Issues

The actual connectivity of mobile peers depends on the network coverage and capacity, as well as the type of peer mobility. In this scenario, we evaluate the case of subway passengers in order to analyze the effects of the intermittent connectivity of peers on resource allocation. For this purpose, we use our dataset[7] of mobile connectivity for the subway system in Paris. The used data show that subway travelers loose and recover network connection for periods that range from several seconds to 5 minutes, maximum. On average, connected periods are 1.5 times larger than the disconnected periods. We randomly select the path *Dugommier* → *Cité Universitaire* for this study of subscriber mobility. By analyzing the complementary cumulative distribution functions of connections and disconnections in the above subway path, we conclude that our traces fit best with an exponential distribution, such as $T_{s_i}^{\text{ON}} = 155.8$ sec and $T_{s_i}^{\text{OFF}} = 96$ sec. Fig. 7 presents the results for selected thresholds.

QoS enforcement is more challenging with intermittent connectivity. For the given parameters it is not possible to achieve response times lower than 22.95 sec due to the overhead in the processing of buffered events. For this threshold, there is QoS violation in some cases but the values were on average only 3.11% different from the thresholds. The resource allocation cost was, on average, \$10.03, \$0.14, and \$0.03, respectively. This result enforces the need for such a solution for resource allocation tunning since a small change in response time threshold (an

[7] `https://github.com/boulouk/mobile-jinqs/tree/master/experiments/metro`

Table 1. Resource synthesis processing time (in sec)

Total number of brokers	Number of brokers in each routing path					
	1	2	3	4	5	6
1	0.03					
2	0.04	0.09				
3	0.06	0.12	0.16			
4	0.08	0.21	0.68	0.99		
5	0.07	0.37	2.14	10.04	14.69	
6	0.10	0.55	5.91	62.41	346.03	489.36

increasing of 8.95% on tolerated response time on this case) can result in a substantial decrease in resource allocation cost (a saving of 99.7% on cost).

5.3 Performance Evaluation

In the previous experiments, we demonstrated the effectiveness of the resource synthesis approach on QoS enforcement. However, we performed this evaluation using a single event routing path. In order to extend this analysis, we measured the approach performance for higher scale scenarios. For this purpose, we analyzed the processing time to get results when taking many routing paths as input. We do this evaluation using the scenario from the previous experiment with always-connected subscribers, and a publishing rate of 23 events/sec. We set $\Delta^{thr} = 10$ sec to make QoS fulfillment possible as the scale increases.

In this experiment, we create synthetic pub/sub topologies by varying $|B| = [1..6]$. By taking paths of different scales (i.e., number of participating brokers) we generate all possible options in the pub/sub system. In the context of our future work, we intend to use specific event routing algorithms (e.g., selective routing or event gossiping [5]) in order to scale our approach (the set of possible end-to-end paths will be reduced). For each scenario, we generate all routing paths with same number of communicating message brokers as follows: we generate all combinations without repetition taking $l = [1..|B|]$ message brokers per path. For each resulting set we then create all possible permutations. Using this strategy, each scenario has a topology with $\binom{|B|}{l} l! = \frac{|B|!}{(|B|-l)!}$ routing paths what allow us to evaluate the scalability of the approach.

We run the synthesis approach for the same input 100 times and collect the average processing time and confidence interval. All executions were performed on a machine with the following configuration: Intel CPU® Core™ i5 2.67GHz, 4GB RAM, Ubuntu 14.04. The average processing time for each scenario is presented in Table 1. Resource synthesis processing time varies from 0.03 sec to 8.16 min. The high processing time in the worst case scenario is due to a high number of routing paths. In order to evaluate the feasibility of the solution, the processing time must be compared with the duration of the other tasks in the pub/sub system deployment. Regarding this, since we do not use previously instantiated VMs, VM startup is the main bottleneck. The reason for this is that VMs have to be instantiated and configured, which includes communication with the cloud provider, VM loading, and installation of software components. Even though we have not conducted any experiments in this regard, there are a number of results in the literature that indicate an overhead from 44.2 sec to 13.5 min

on VM startup in public clouds [22]. We argue that the overhead imposed by the resource synthesis is acceptable, especially if we consider the achieved benefits.

6 Conclusions and Future Work

In this paper, we propose a new approach for resource allocation for IoT applications in the Cloud. Such applications run over pub/sub systems and their *things* can be intermittently disconnected. Our approach can be used as a tool by pub/sub system designers to allocate resources for QoS enforcement. We have evaluated the proposed solution using scenarios based on probability distributions of the events, as well as scenarios based on connectivity parameters derived from real traces. QoS enforcement was achieved in almost all scenarios, which showed only 3.11% of QoS violation in some cases. We conclude that resource demand increases with higher arrival rates but also due to peer disconnections. Consequently, the cost of deploying the system is also affected by these parameters. However, we have shown that our approach can help reasoning about efficient resource allocation in a variety of scenarios, saving up to 99.7% on resource allocation cost. Finally, our approach has a feasibly processing time, as corroborated by the experiments.

As future work, we intend to identify the limits for system saturation, given the available resources. We also aim to carry out a more comprehensive performance evaluation using real pub/sub protocols. As further unfolding of the work, we aim to extend this proposal with support for QoS enforcement when using multiple regions.

References

1. Aarts, E., Korst, J.: Simulated annealing and Boltzmann machines. New York, NY; John Wiley and Sons Inc. (1988)
2. Amazon: Amazon Web Services - Broad & Deep Core Cloud Infrastructure Services (2018), `http://aws.amazon.com`, [Accessed: 28-Mar-2018]
3. Apache Kafka: (2018), `http://kafka.apache.org/`, [Accessed: 28-Mar-2018]
4. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer networks pp. 2787–2805 (2010)
5. B., R., Q., L., V., A.: Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey. Tech. rep., Universita di Roma La Sapienza (2005)
6. Banks, A., Gupta, R.: MQTT Version 3.1. 1. OASIS standard (2014)
7. Barazzutti, R., Heinze, T., Martin, A., Onica, E., Felber, P., Fetzer, C., Jerzak, Z., Pasin, M., Rivière, E.: Elastic scaling of a high-throughput content-based publish/subscribe engine. In: IEEE 34th ICDCS. pp. 567–576. IEEE (2014)
8. Botta, A., De Donato, W., Persico, V., Pescapé, A.: Integration of Cloud Computing and IoT: a Survey. Future Generation Computer Systems 56, 684–700 (2016)
9. Bouloukakis, G., Georgantas, N., Kattepur, A., Issarny, V.: Timeliness evaluation of intermittent mobile connectivity over pub/sub systems. In: Proc. of the 8th ACM/SPEC on Int. Conf. on Performance Engineering. pp. 275–286. ACM (2017)

10. Bouloukakis, G., Moscholios, I., Georgantas, N., Issarny, V.: Performance modeling of the middleware overlay infrastructure of mobile things. In: Communications (ICC), 2017 IEEE International Conference on. pp. 1–6. IEEE (2017)
11. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience (2011)
12. Carvalho, N., Araujo, F., Rodrigues, L.: Scalable qos-based event routing in publish-subscribe systems. In: Network Computing and Applications, Fourth IEEE International Symposium on. pp. 101–108. IEEE (2005)
13. Collberg, C., Myles, G., Stepp, M.: An empirical study of java bytecode programs. Software: Practice and Experience pp. 581–641 (2007)
14. Corsaro, A., Querzoni, L., Scipioni, S., Piergiovanni, S.T., Virgillito, A.: Quality of service in publish/subscribe middleware. Global Data Management (2006)
15. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.: The many faces of publish/subscribe. ACM Computing Surveys pp. 114–131 (2003)
16. Gascon-Samson, J., Garcia, F., Kemme, B., Kienzle, J.: Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. In: IEEE ICDCS. pp. 486–496 (2015)
17. Google: Compute Engine (2018), `https://cloud.google.com/compute`, [Accessed: 28-Mar-2018]
18. Gross, D., Shortle, J., Thompson, J., Harris, C.: Fundamentals of queueing theory. John Wiley & Sons (2008)
19. Hoffert, J., Schmidt, D.C., Gokhale, A.: Adapting distributed real-time and embedded pub/sub middleware for cloud computing environments. In: IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing. Springer (2010)
20. Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc. (1984)
21. Lydia, L., Raj, B., Craig, L., Dennis, S.: Magic Quadrant for Cloud Infrastructure as a Service, Worldwide (2017), `https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519`, [Accessed: 28-Mar-2018]
22. Mao, M., Humphrey, M.: A Performance Study on the VM Startup Time in the Cloud. In: IEEE CLOUD. pp. 423–430 (2012)
23. Microsoft: Microsoft Azure (2018), `https://azure.microsoft.com`, [Accessed: 28-Mar-2018]
24. Moser, M., Jokanovic, D.P., Shiratori, N.: An algorithm for the multidimensional multiple-choice knapsack problem. IEICE 80(3), 582–589 (1997)
25. Nguyen, P., Nahrstedt, K.: Resource management for elastic publish subscribe systems: A performance modeling-based approach. In: IEEE CLOUD (2016)
26. Oracle: JMS Specifications (2018), `http://www.oracle.com/technetwork/java/jms/index.html`, [Accessed: 28-Mar-2018]
27. Pivotal: RabbitMQ (2018), `https://www.rabbitmq.com/`, [Accessed: 28-Mar-2018]
28. Setty, V., Vitenberg, R., Kreitz, G., Urdaneta, G., van Steen, M.: Cost-effective resource allocation for deploying pub/sub on cloud. In: IEEE ICDCS (2014)
29. Shi, W., Dustdar, S.: The promise of edge computing. Computer pp. 78–81 (2016)
30. Standard, O.: Oasis advanced message queuing protocol (amqp) version 1.0.
31. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM TWEB (2007)
32. Zhao, W., Peng, Y., Xie, F., Dai, Z.: Modeling and simulation of cloud computing: A review. In: IEEE APCloudCC. pp. 20–24 (2012)