MultiPub: Latency and Cost-Aware Global-Scale Cloud Publish/Subscribe

Julien Gascon-Samson, Jörg Kienzle, Bettina Kemme
School of Computer Science, McGill University
Montreal, QC H3A 0E9, Canada
Email: Julien.Gascon-Samson@cs.mcgill.ca,{Joerg.Kienzle,Bettina.Kemme}@mcgill.ca

***Abstract*—Topic-based pub/sub is a widely used communication mechanism in distributed systems for targeted information dissemination between loosely coupled entities. To scale dynamically depending on the current communication demands, pub/services can be conveniently deployed in the cloud. To provide fast dissemination, the service can be distributed across multiple cloud regions. The architectural design and run-time deployment of such a middleware is tricky, though, as it can have a significant effect on communication latency and cloud-based cost. In this paper, we propose MultiPub, a flexible pub/sub middleware for latency-constrained, world-wide distributed applications that dynamically reconfigures the communication layer to ensure a predefined maximum latency for publication dissemination while minimizing cloud-based costs. This is achieved by routing publications either through a single or across multiple cloud regions. We demonstrate the effectiveness of MultiPub by presenting a set of experiments that report on the achieved communication latency and cost savings compared to traditional approaches, as well as a performance evaluation.**

I. Introduction

The publish/subscribe paradigm (pub/sub) and design pattern allows for decoupling content producers (publishers) from content consumers (subscribers) in an elegant, yet simple way. In the most common, topic-based pub/sub model, subscribers declare their interest by subscribing to topics. Publishers send publication messages by tagging each publication with the topic it belongs to. The publication is then disseminated to all subscribers that are interested in the topic [14]. All communication is done through a pub/sub middleware to which both publishers and subscribers connect.

Due to the simple yet flexible model, topic-based pub/sub systems are used across a wide range of small to large-scale applications, such as social media, chat/group systems, weather and traffic alert systems, among many others. Well-known large-scale applications of topic-based pub/sub are push-notification systems for mobile devices such Google Cloud Messaging, where applications register as subscribers to receive notifications, and where publishers push relevant notifications towards applications [2]. Large-scale multiplayer online games also make use of pub/sub for efficient dissemination of game-related updates to players [16], [20], [12].

Many of these applications have stringent latency requirements. Games usually require very tight latency bounds in order to provide an enjoyable and immersive experience for gamers from less than 150 ms for first-person-shooter games to an upper bound of 500 ms for role-playing games [9]. Push-notification systems used for alerts and monitoring applications also have tight latency requirements. Any grouping service with audio- or video-dissemination also falls into this category. Meeting the needs of such latency-constrained applications with participants in several regions of the world can be challenging.

Deploying the pub/sub infrastructure in the cloud brings many advantages, such as virtually unlimited scalability and optimized connections to the Internet backbone, which can lead to better delivery times. Also attractive for latency-constrained applications is the fact that cloud providers usually offer clouds in different regions, so that users can connect to the service instance in their region leading to fast client/server interaction. However, cloud providers charge on a per-use basis, whereby outgoing bandwidth cost is likely to be the most costly factor for a pub/sub dissemination service. This and the fact that costs can vary significantly from region to region require a careful look at how and where to deploy a pub/sub service across cloud regions such as to both provide performance to the users and keep cost at bay.

In this paper, we propose MultiPub, a fully dynamic global-scale topic-based pub/sub middleware, tailored towards applications which require meeting strict delivery time bounds. MultiPub allows the user to set timing constraints on a per-topic basis, and ensures that such constraints are respected whenever possible. Depending on client locations, delivery bounds and cloud costs, a given topic can be managed by server(s) within a single or multiple cloud regions. MultiPub automatically finds the best configuration in terms of costs that does not violate delivery guarantees, and reconfigures whenever conditions change.

In particular, our paper provides the following contributions:

- We provide a ready-to-use dynamic global-scale topic-based pub/sub middleware that can be deployed as a service in the cloud.
- Our approach takes advantage of cloud providers that offer clouds in geographically dispersed regions and considers their specific characteristics (bandwidth costs, latencies), in order to generate an optimal configuration in terms of assigning regions to serve topics and clients. Two different message routing approaches are supported.
- Our system automatically collects and analyzes real-time measurements from all nodes and continuously reconfigures itself whenever a more appropriate configuration is found. The process is entirely transparent to users.
- We have built a complete simulation package to extensively evaluate our model under different scenarios.
- We have gathered real latency measurements towards and between all regions of the Amazon EC2 cloud [3]. Combined with the King dataset [17], we derive realistic client latency values to conduct realistic experiments.

R	Region	Location	$EC2	$Inet
R_1	us-east-1	N. Virginia	0.02	0.09
R_2	us-west-1	N. California	0.02	0.09
R_3	us-west-2	Oregon	0.02	0.09
R_4	eu-west-1	Ireland	0.02	0.09
R_5	eu-central-1	Frankfurt	0.02	0.09
R_6	ap-northeast-1	Tokyo	0.09	0.14
R_7	ap-northeast-2	Seoul	0.08	0.126
R_8	ap-southeast-1	Singapore	0.09	0.12
R_9	ap-southeast-2	Sydney	0.14	0.14
R_{10}	sa-east-1	Sao Paulo	0.16	0.25

Table I: EC2 Outgoing Bandwidth Costs

II. MULTIPUB MODEL

MultiPub is a topic-based cloud publish/subscribe middleware that is tailored for latency-constrained, world-scale applications. MultiPub servers can be installed in as many cloud regions as necessary to serve clients. Given a set of regions, and a topic with its publishers and subscribers, MultiPub automatically determines which regions should serve the topic considering that publishers and subscribers are located worldwide and have varying latencies towards the different regions. The decision takes into account two factors. First, delivery times should be kept under a user-defined threshold. Second, since the use of the cloud incurs costs on a pay-per-use basis, MultiPub chooses, among the configurations that fulfill the delivery constraints, the configuration that is the cheapest in terms of cloud costs. Table I shows the 10 regions offered by Amazon EC2. Throughout this paper, we use this region setup as an example for both latency as well as cost calculations.

A. Delivery Constraints vs. Cost Minimization

With MultiPub, a delivery time constraint can be specified for each topic T. The constraint specifies the maximum allowed delivery time (max_T) for a ratio of all publications received across T ($ratio_T$). For instance, $max_T = 200$ and $ratio_T = 95$ mean that 95% of all messages sent on T should be delivered within 200 ms or less. MultiPub makes sure that this constraint is respected whenever possible. However, in some cases, it might be unrealistic, as some clients might experience very high latencies due to the use of bad connections, or the requested max_T threshold might simply be too low. If the constraint cannot be met, then MultiPub finds the most latency-minimizing configuration. A configuration for a topic T defines the regions which serve the topic, as well as for any publisher or subscriber client, to which server(s) they connect. We will discuss the configuration options shortly.

The MultiPub cost model only considers bandwidth-related costs as this is by far the dominating factor. Given the simplicity of topic-based matching the costs related to message dissemination are much higher than CPU costs. In current cloud infrastructures, cloud inbound bandwidth is typically free, while there are different costs associated with outgoing bandwidth towards other cloud regions and outgoing bandwidth to external clients. As the baseline for this paper, Table I details the costs in terms of outgoing bandwidth of the various EC2 regions. Column `$EC2` gives the costs of 1GB of outgoing data sent towards another EC2 region, while column `$Inet` lists the costs of 1GB of data sent towards any node on the Internet. We observe that the outgoing costs in some regions (Asia and South America) are very expensive compared to others. Thus, it might be worth avoiding to route topics through servers in regions with expensive bandwidth costs if the delivery constraints allow for such optimization.

B. Configuration Options

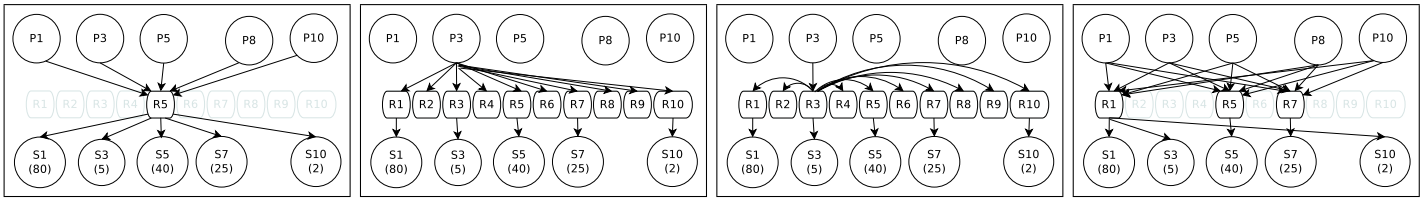
Supporting publishers and subscribers in different cloud regions, while ensuring that latency constraints are respected, can be challenging. Figure 1 presents different approaches combining several cloud regions that can be used to process publications on a given topic T. Assuming the 10 EC2 regions and topic T with 5 publishers, one each close to the regions R_1, R_3, R_5, R_8, and R_{10} respectively, and 5 groups of subscribers, each group being close to one of these five regions, and having 80, 5, 40, 25, and 2 subscribers respectively.

1) One Region: In the simplest case (shown in figure 1a), the pub/sub middleware in only one region is in charge of publications on topic T (for instance, region R_5, in Frankfurt). In this scenario, all publishers and subscribers for this topic connect to the service in this region. With this setup, some clients will experience very high latencies due to fact that some publications will travel for long and/or cross ocean distances twice. For instance, upon P_1 (close to region R_1, i.e., North Virginia) publishing to T, the publication needs to travel from P_1's location to the cloud region R_5 (long distance), where the pub/sub service then sends the publications to all subscribers. All subscribers except of group S_5 will receive the publication only after a long delay as they are far away from R_5. The advantage of this approach is, however, that is offers a very cheap solution, as R_5 is one of low-cost regions.

2) All Regions: This approach involves statically having pub/sub servers in all cloud regions handle T. This scheme allows for minimizing delivery times, since each subscriber automatically uses its closest, local cloud region (figures 1b and 1c). With the *direct delivery* approach (figure 1b), upon publishing, all publishers send all publications towards all regions (note that the figure only shows publications from one publisher to simplify). One problem of this approach is that all publishers must send their publications to all regions, which can be cumbersome, in particular as outgoing bandwidth might be a limiting factor for some publishers.

Routed delivery (figure 1c) is an alternative scheme where publishers only send towards their local, closest region (again, only one publisher is shown in the figure). The local region then forwards the publication to the pub/sub service in all other regions. This scheme solves the issue of publishers publishing towards all regions, at the expense of increased delivery costs due to the additional outgoing, inter-region cloud bandwidth costs. Although the cost is generally smaller then sending to clients, it can still sum up quickly.

At first view it might appear that using the direct delivery approach will always yield lower latencies than routed delivery as messages always only travel two hops (from publisher to the service, and from there to the subscribers) while routed delivery has one further redirection (from cloud region to another cloud region). However, as inter-cloud links are often more optimized, the actual latencies can vary significantly, and thus direct delivery might have lower latency in some



(a) One Region (b) All Regions / Direct (c) All Regions / Routed (d) MultiPub / Direct Delivery

Figure 1: Publication Delivery Approaches

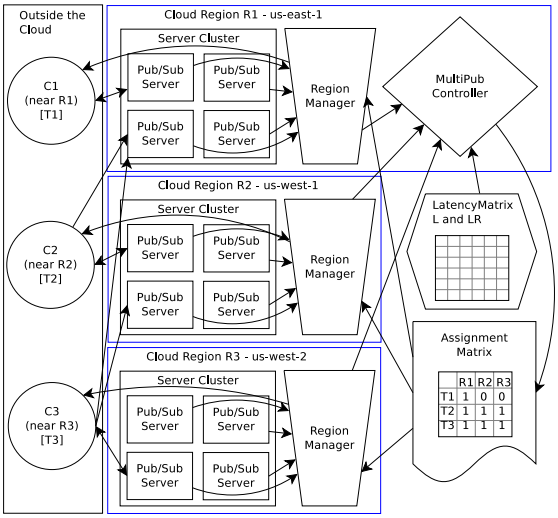


Figure 2: MultiPub Architecture

configuration while routed delivery can have lower latency in other configurations. Section V-D describes an experiment that compares direct against routed delivery. An obvious drawback of the *all regions* approach is that this scheme forces all regions to be used for topic T despite some regions having very few or no clients, thus consuming additional potentially unnecessary resources, and being potentially more expensive due to the usage of regions exhibiting higher costs.

3) MultiPub: MultiPub finds the best combination of cloud regions to use among all possible regions for topic T. MultiPub also determines whether direct or routed delivery should be used. Figure 1d shows an example of MultiPub with only 3 regions selected for topic T out of the 10 possible regions, using the direct delivery approach. Since there are very few subscribers in regions R_3 and R_{10}, MultiPub chooses to ignore these regions. Subscribers close to these two regions are assigned to region R_1 instead, since this region happens to be their second closest region. As this was the case for the "all regions" approach, MultiPub can also opt for routed delivery (not shown in the figure). In that case, publisher P_3 would publish towards its closest available region (R_1 in Virginia, since R_3 in Oregon has not been selected by MultiPub for topic T). The MultiPub instance installed in R_1 forwards the publication to R_5 and R_8.

III. System Architecture and Modelling

In this section, we first holistically describe the architecture of MultiPub (depicted in Figure 2). We then describe the pub/sub model more formally.

A. Architecture

1) Server Clusters: Each cloud region has an instance of the MultiPub service. The pub/sub matching itself can be performed by one server or a set of servers. MultiPub uses internally Dynamoth [15], a pub/sub service that automatically and dynamically provisions the number of servers needed to handle the current load on all topics installed. However, in principle, we could use any scalable pub/sub platform that can be deployed in a single cloud. We consider supporting multiple servers per region as an orthogonal problem since intra-region scalability can be managed locally, on a per-region basis, without altering the way in which MultiPub behaves. To simplify, we assume there is a single server per region, and use the terms "service" and "server" interchangeably.

2) Assigning Regions to Topics: MultiPub allows topics to be handled by one or more regions. The mapping of topics to regions is expressed as a bit matrix, referred to as *assignment matrix*, with the columns being the regions and the rows being the topics. The row for topic T contains a 1 for each column representing a region that serves the topic, and a 0 for regions that do not serve the topic.

3) Region Managers: Each region has a *region manager* component that collects real-time data for every topic T maintained by the region, including the list of all publishers who publish to T, the list of subscribers who have subscribed to T, as well as the number of messages and their sizes in bytes sent by each publisher over T. Data is collected throughout a collection interval, and then sent to the MultiPub Controller which can then determine overall delivery times and bandwidth costs and make necessary adjustments.

4) MultiPub Controller: The *MultiPub controller* is installed in one of the regions. It aggregates the data received from the region managers. Furthermore, it maintains for each topic T, the delivery constraint $< ratio_T, max_T >$ (where the $ratio_T$ percentile of messages must be delivered to subscribers within a time bound of max_T). Finally, it keeps track of the latencies between every client C in the pub/sub system (publisher or subscriber) and each of the cloud regions, as well as the latency between each pair of cloud regions.

5) Deploying a New Configuration: Using the latency values, combined with the latest information about each topic collected by the region managers, the MultiPub controller continuously recomputes an optimal configuration for topic T. If a better configuration is found, it is sent in the form of a bit vector to the region managers which then incorporate them into their assignment matrix. The new configuration also has to be sent to affected clients of this topic. This process is handled by the region managers: upon a configuration change being requested for a given topic T, the region manager in

each region R informs the clients (subscribers and publishers) that were *closest* to R latency-wise (in the old configuration) that a new configuration must be used for T.

Then, if a region was added for T, and the new region is closer to a given subscriber than any previous region of this topic, then the subscriber has to connect to the new region. Correspondingly, if a region was removed, then the subscribers that were currently connected to this region must reconnect to the next closest region. Similar holds for publishers.

For example, assuming a scenario with a topic T with 10 publishers and 10 subscribers in North America and 10 publishers and 10 subscribers in Europe, and only a server in Region R_1 assigned to T. The MultiPub controller determines that subscribers in Europe experience delivery times that are over the threshold for a significant portion of the publications that they receive (more precisely, for all publications sent from a publisher in Europe and received by a subscriber in Europe, as these messages cross the Atlantic twice). The controller then decides that T should now map to two regions: R_1 (us-east-1) and R_5 (eu-central-1) with a direct delivery approach in order to meet delivery constraints. All 20 publishers need to receive the new configuration so that they now send their publications to both regions, as well as the 10 subscribers in Europe as they have to resubscribe to the European region R_5. With this, any message now crosses the Atlantic at most once.

B. Publishers, Subscribers, Regions and Publications

Assuming a particular configuration for a topic T, consisting of publishers, subscribers and a set of regions containing the MultiPub service that handle T, we show how delivery times and the bandwidth costs over a given time period can be formally calculated.

We consider a total of N_R^{total} regions. Taking topic T, we denote with N_P the number of publishers for T, with N_S the number of subscribers to T and with N_R the number of regions that are assigned to T. $\mathbb{S} = \{S_1, \ldots, S_{N_S}\}$ is the set of subscribers for topic T, $\mathbb{P} = \{P_1, \ldots, P_{N_P}\}$ is the set of publishers for T, and $\mathbb{R} = \{R_1, \ldots, R_{N_R}\}$ is the set of regions serving T.

As discussed before, we consider both direct delivery where a publisher P sends a message to all regions in $\mathbb{R}$, and routed delivery where it sends it to only one region, denoted as R^P, who then forwards it to the other regions in $\mathbb{R}$. R^P is the region that is the closest (latency-wise) to P. All subscribers of topic T connect to only one region $R^S \in \mathbb{R}$, namely the closest. We denote with $\mathbb{S}^R \subset \mathbb{S}$ the subset of subscribers that use region R to receive publications on topic T, and with N_S^R the number of subscribers that subscribe to topic T on region R. Therefore, $N_S = \sum_{R \in \mathbb{R}} N_S^R$.

Given a publication message M sent by publisher P on a given topic T, M is sent to each of the regions $R \in \mathbb{R}$ (either by P directly or through R^P) and then each region R forwards it to all the N_S^R subscribers, leading to N_R messages towards all cloud regions handling T and a total of N_S messages towards all subscribers.

C. Latency Model

A subscriber S to topic T connects to $R^S \in \mathbb{R}$, which is the closest region. Similar holds for publishers in case of routed delivery. To determine the qualifying region, we maintain a latency matrix L, where each row represents a client C (either a publisher or a subscriber) and each column a cloud region R. There is a total of N_R^{total} columns, that is, also columns for regions that do currently not serve topic T. Entry L_{CR} indicates the expected one-way latency (message delivery time) between client C and region R (either direction). Thus, a subscriber S (publisher P) connects to $R^S \in \mathbb{R}$ ($R^P \in \mathbb{R}$) with the smallest $L_{SR^S}(L_{PR^P})$ value among all regions in $\mathbb{R}$. We assume L_{CR} to remain constant, but our model still holds if the value is updated over time at an infrequent rate. For instance, the infrastructure can monitor latency changes between every client C and every available region R and update L_{CR} accordingly.

We also define a further matrix (L^R), which represents one-way latencies between pairs of cloud regions. $L^R_{R_iR_j}$ denotes the latency between region R_i and region R_j. Obviously $L^R_{R_iR_i} = 0$. Results for the 10 regions of the Amazon EC2 cloud were determined and are presented in section V-A1. As mentioned previously, the MultiPub controller optimizes the placement of topics on cloud regions as well as the delivery approach by taking L and L^R into consideration.

D. Publication Delivery Time

The total delivery time $D(M^{PS})$ of any given publication M sent from publisher P on topic T towards subscriber S depends on whether direct or routed delivery is used. For simplification purposes, we assume that the processing time of message M at any node is negligible which we believe is reasonable given that finding the list of subscribers for a given topic can be implemented as a simple lookup operation.

1) Expected Direct Delivery Time: Publisher P directly sends its publication to all regions $R \in \mathbb{R}$, which then forward it to their local subscribers. Thus, the delivery includes two hops and is calculated for subscriber S connected to the closest region R^S as

$$D_{Direct}\left(M^{PS}\right) = L_{PR^S} + L_{R^SS} \tag{1}$$

2) Expected Routed Delivery Time: Publisher P sends its publication M towards its local region R^P, i.e., the one with minimal latency. R^P then forwards M to all other $R \in \mathbb{R}$. Each region then forwards M to their local subscribers. Thus, the delivery includes either two hops (for the subscribers with $R^S = R^P$) or three hops, and can be calculated as

$$D_{Routed}\left(M^{PS}\right) = L_{PR^S} + L^R_{R^PR^S} + L_{R^SS} \tag{2}$$

E. Publish/Subscribe Cost Model

As mentioned, the MultiPub cost model only considers bandwidth-related costs. In the following, we designate with $\alpha(R)$ the cost per outgoing byte from region R towards a different region (derived from column `$EC2` in table I), and with $\beta(R)$ the cost per outgoing byte towards any client-subscriber of R (derived from column `$Inet`).

In order to calculate the overall bandwidth costs for our approaches, we need some further information about the number of messages per collection interval and their size. Thus, for each publisher P we need to know the number of messages N_M^P sent by P to topic T, and for each of these messages M_1^P,M_2^P, ... $M_{N_M^P}$ its size $\Omega(M_j^P)$ in bytes.

With this, we can calculate the total costs Z_{Direct} for topic T using the direct delivery approach:

$$Z_{Direct} = \sum_{k=1}^{N_P} \sum_{j=1}^{N_M^P} \sum_{i=1}^{N_R} N_S^{R_i} \times \Omega(M_j^{P_k}) \times \beta(R_i) \quad (3)$$

That is, for each publisher P, for each of its messages M_j^P and for each of the regions R_i serving topic T, the outgoing bandwidth is the size of the message multiplied by the number of subscribers using region R_i multiplied by the bandwidth costs per byte.

The total costs Z_{Routed} for topic T using the routed delivery approach have to additionally consider that the region local to a publisher P forwards the message to all other regions serving topic T (of which there are $N_R - 1$):

$$Z_{Routed} = Z_{Direct} + \sum_{k=1}^{N_P} \sum_{j=1}^{N_M^P} (N_R - 1) \times \Omega(M_j^{P_k}) \times \alpha(R^P) \quad (4)$$

IV. Optimization Problem

For each topic T, the MultiPub controller determines on a regular basis the optimal configuration given the topic's delivery constraint $< ratio_T, max_T >$, the publishers and subscribers of T in the last observation interval, and the number and size of the messages sent by the publishers in that observation interval. To this aim, the controller has to consider every possible assignment of the topic to the existing regions. An assignment for T can be encoded as a bit vector: the bit for each region can either be set (topic assigned to the region) or not set. As this represents one row of the assignment matrix, we refer to it as assignment vector V for topic T. Thus, given that there are a total of N_R^{total} regions, then there are $2^{N_R^{total}} - 1$ possible assignments (it is not possible that all region bits are zero). Furthermore, if at least two bits are set, then there is the option of either direct or routed delivery. Thus, there are a total of $2 \times (2^{N_R^{total}} - 1)$ -N_R^{total} possible configurations. Although this is exponential in the number of regions, one has to be aware that the number of regions is fairly small. For each of these configurations, MultiPub calculates what would have been the delivery times for all messages sent in the last observation interval, in order to determine if the configuration is a suitable one; i.e., if it would have fulfilled the topic's delivery constraint $< ratio_T, max_T >$. Then, among all *suitable* configurations, it determines the best one.

A. Checking for Delivery Constraint

Given a possible configuration C for topic T (possible value for assignment vector V and either direct or routed delivery), MultiPub first determines for each subscriber $S \in \mathbb{S}$ (and in case of routed delivery publisher $P \in \mathbb{P}$) the closest region R^S (R^P) for which the bit is set in V. With this, it calculates for each publisher P and subscriber S the latency $D(M^{PS})$ observed for sending a publication from P to S as expressed in Equations 1 and 2 of Section III-D for direct and routed delivery, respectively.

From there, MultiPub creates a list $\mathbb{D}_\mathbb{C}$ which contains for each publisher P and for each of its messages M_1^P,M_2^P, ... $M_{N_M^P}$ sent in the observation interval, the delivery times (either $D_{Direct}(M_i^{PS})$ or $D_{Routed}(M_i^{PS})$). $\mathbb{D}_\mathbb{C}$ is sorted by delivery time with the shortest delivery time first. The cardinality $|\mathbb{D}_\mathbb{C}|$ is the total number of messages sent between publishers and subscribers, i.e., the total sum of all messages sent by all publishers multiplied by the number of subscribers $N_S \times \sum_{k=1}^{N_P} N_M^{P_k}$. The delivery constraint for topic T requires that a fraction of $ratio_T$ messages be delivered in at most max_T time. This can be translated into checking whether the n^T-ieth delivery time in the sorted list $\mathbb{D}_\mathbb{C}$ is still at most max_T where:

$$n^T = \left\lceil \frac{ratio_T}{100} \times |\mathbb{D}_\mathbb{C}| \right\rceil \quad (5)$$

We refer to this n^T-ieth delivery time as delivery time percentile $\mathring{D}_C$ for configuration C and it has to be lower or equal to max_T. The following constraint has to be fulfilled for C to be further considered.

$$\mathring{D}_C = \mathbb{D}_\mathbb{C}\left[n^T\right] \leq max_T \quad (6)$$

B. Determining the Optimal Solution

Given a possible configuration C that fulfills the delivery constraint, the bandwidth cost Z_C is then calculated according to equations 3 and 4 of Section III-E. After having determined the delivery time percentile and bandwidth costs for each configuration C, we sort the configurations by costs. Then we take the configuration with the lowest cost that fulfills the delivery constraint. If there are several configurations that have the same lowest cost, we choose the one that has the lowest delivery time percentile. If there are several that also have the same delivery time percentile then we choose the one that uses the least amount of servers. In contrast, if no configuration fulfills the delivery constraint we take the one with the lowest delivery time percentile irrespectively of its cost.

C. Independence of Topics

MultiPub minimizes costs for all topics, while respecting all constraints (whenever possible) for every topic. Minimizing the costs for every topic leads to a global minimization of the overall costs. Since there is no global constraint, or inter-topic constraints, all topics can then be considered as independent. Therefore, we have a distinct optimization problem for each topic. In other words, the outcome of optimizing one single topic will not impact the optimization of any other topic.

D. Handling Clients experiencing High Latencies

It might happen that some clients temporarily experience higher latencies, which might put them at a disadvantage, as all of their latencies could potentially fall above the requested delivery time percentile over a given topic T. As a mitigation mechanism, the MultiPub controller periodically scans for such clients. Upon detecting one such client C, then the controller verifies whether forcing the addition of any given cloud region to the currently selected list of regions for T would allow for the latency needs of C to be met (if possible), or to be improved significantly. Should that be the case, then the region is added to the list of regions, and when it becomes no longer needed, it is removed.

V. Experimental Validation

In this section, we analyze the behaviour and performance of MultiPub under a wide range of configurations. Unfortunately, running cross-cloud experiments is costly due to VM rental cost and cross-cloud outgoing bandwidth costs. Therefore, we opted for a simulation-based approach.

A. Determining Latencies for Simulation

Since our formal modelling depends on the inter-cloud latency matrix L^R and on the client-to-cloud latency matrix L, we need to generate appropriate and realistic latency values. In the following subsections, we describe the methodology that we employed to generate reliable values for L and L^R.

1) Inter-Cloud Latencies (L^R): As explained previously, we have access to servers in all regions of the EC2 cloud (and we could have access to other clouds as well); therefore, we are able to measure latencies between machines in all pairs of regions and generate the latency matrix L^R. To do so, we ran one $t2.micro$ VM in each of the 10 Amazon EC2 cloud regions, and we repeatedly measured the *ping* towards each other region (100 times), which we averaged and which we divided by 2 to obtain the single-trip time.

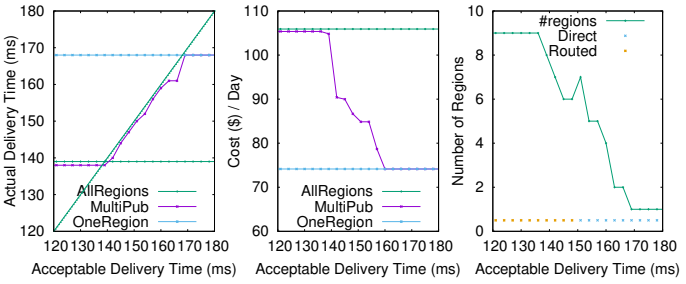
2) Client to Cloud Region Latencies (L): To determine realistic latency measurements between clients and cloud regions, we used the publicly available King dataset [17], which contains latency measurements for communications between over 1800 world-wide geo-distributed DNS servers. We setup a VM in the 10 EC2 regions and attempted to *ping* each of the 1800 DNS servers. We were able to obtain latency measurements from 700 of them, which allowed us to build the latency matrix L. Since the DNS servers are distributed across the globe, we can use their latency values to estimate client-cloud connection latencies from most places in the world.

B. System Implementation and Experimental Setup

We implemented a full simulation package in Python 3.5.1. It has access to the measured real-world latencies stored in the matrices L and L^R, as well as the outgoing bandwidth costs (towards the Internet and towards another EC2 region) for each of the EC2 regions (table I). The simulator can run simulations with any number of topics. For each topic, the number of publishers and subscribers can be specified. Furthermore, for each publisher, a specific publication rate and publication size must be configured, as well as which of the Amazon EC2 regions the publisher machine should be geographically closest to. Finally, for any given topic T, the upper bound in terms of maximum acceptable delivery time (max_T) and the ratio/percentile $ratio_T$ of all delivery time measurements that should be below max_T must be specified. We ran several simulation experiments, as well as a runtime analysis, which are described in the next subsections.

C. Experiment 1 - MultiPub vs. Other Approaches

The goal of this experiment is to compare MultiPub against other approaches in a global setting. We simulated one topic T with 100 globally-distributed publishers and subscribers, where always 10 publishers and 10 subscribers are located close to one of the EC2 regions. Each publisher publishes on average once per second (message size of 1 KByte).



(a) Delivery Time (b) Costs (c) # Regions

Figure 3: Comparison of MultiPub against other approaches

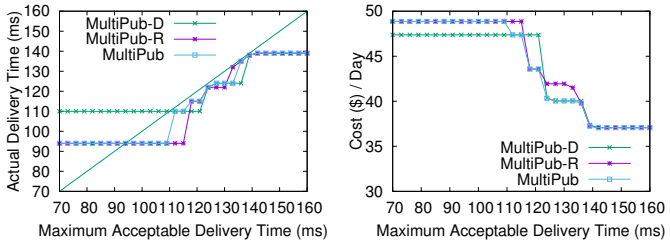
We compare MultiPub, where we vary the delivery time bound max_T between 100ms and 200ms, against a) the "All Regions (Routed Delivery)" model (see section II-B2), which should yield the fastest results because publishers and subscribers use the region that minimizes their delivery time, and b) the "One Region" model (see section II-B1), which should yield the cheapest results, because the publishers and subscribers favour the region that minimizes costs. For all simulations, we set the delivery time guarantee ratio to 75%.

The simulation results are shown in Figure 3. We observe in figure 3a that the "All Region" approach is able to meet a delivery time bound of 140ms, while the "One Region" approach is able to meet a delivery time bound of 168ms. The MultiPub approach is capable of achieving the same, fast delivery time as the "All Regions" approach when $max_T \leq 140ms$. For higher values of max_T, the actual delivery time increases, but remains under max_T until 168ms. Starting at 168ms, MultiPub aligns itself to the "One Region" approach. Figure 3b depicts the cloud cost calculated as if the test workload had run for a full day on the real cloud. We observe that the fast "All Region" approach is expensive ($107/day), whereas the "One Region" approach is 28% cheaper ($77/day). MultiPub selects the most cost-efficient approach that still meets the target delivery time. For $max_T \leq 148ms$ costs are as high as the "All Region" approach. Between 140ms and 168ms, MultiPub finds a wide range of intermediate configurations that meet the delivery time constraints but use less region servers. This is also visualized in Figure 3c, which plots the number of regions used by MultiPub, and whether publications are routed between cloud servers or only direct communication is used. Since inter-cloud links are generally faster, MultiPub favours routed delivery even if it incurs additional forwarding costs. Eventually, MultiPub opts for direct delivery. Finally, for $max_T \geq 168ms$, only one region is used, i.e., the cheapest one that minimizes delivery time.

To summarize, this experiment demonstrates that MultiPub is able to generate costs savings, which in this specific experiment reached up to 28%, while still respecting delivery time constraints, for a single topic and over one day.

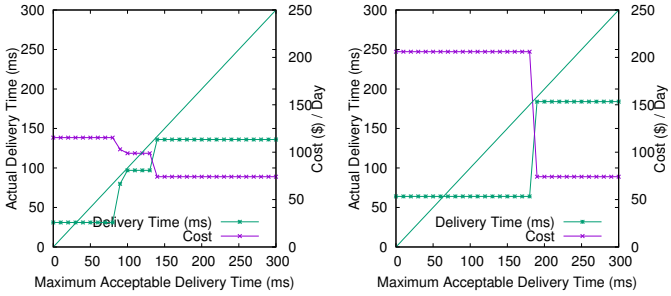
D. Experiment 2 - Direct vs. Routed Delivery

The goal of this experiment is to show that MultiPub is able to exploit direct and routed delivery to reduce delivery times and/or reduce costs. We deployed one topic T with 100 publishers and 25 subscribers in Asia, and 25 subscribers in the USA. Cloud costs are again given for a 1-day period.



(a) Delivery Time (b) Costs

Figure 4: Comparison of Direct and Routed Delivery



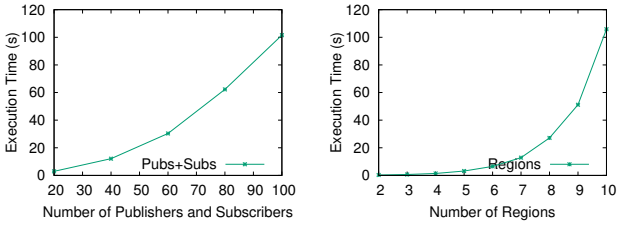
(a) Asia (b) South America

Figure 5: Localized Pub/Sub Delivery across different regions

The delivery time guarantee ratio was set to 75%. We ran 3 separate simulations, one with standard MultiPub, one where we allowed the *controller* to consider direct delivery only (MultiPub-D), and one where the *controller* had to use routed delivery (MultiPub-R). Figure 4a shows that the minimum reachable delivery time with "MultiPub-D" is 110ms, whereas it is 94ms with "MultiPub-R" due to the use of optimized inter-cloud links. Therefore, for $max_T \leq 110ms$, MultiPub uses routed delivery in order to obey the delivery time constraint, despite the costs being higher due to the extra inter-cloud communication (see Figure 4b). For max_T values between 110ms and 138ms, MultiPub selects the best approach that minimizes costs depending on the desired delivery time bound. For values of $max_T \geq 138$, MultiPub chooses the direct delivery approach with only one server, located in the least expensive region that minimizes delivery time.

E. Experiment 3 - Localized Pub/Sub Delivery

In some contexts, publishers and subscribers are local to a region. For instance, in the context of a large-scale online game, players might decide to play against local players. In such a context, the straightforward approach is to deploy the relevant topics only in the local geographical region where the clients are located. However, this setting, while yielding the fastest delivery, is not necessarily the cheapest. This experiment demonstrates that while MultiPub is designed with global-scale publish/subscribe systems in mind, it can also be useful in regional-scale scenarios to optimize costs. We ran the same experiment for two different regions with relatively expensive costs: Asia (EC2 region *ap-northeast-1*) and South America (EC2 region *sa-east-1*). For each experiment on one of the regions R, 100 publishers and 100 subscribers were selected so that they were closest from a latency point of view to region R. The delivery time guarantee ratio was set to 95%.



(a) Publishers and Subscribers (b) Number of Regions

Figure 6: Runtime Analysis

In the Asia experiment (figure 5a), delivery times of 35ms can be achieved using the cloud region in Tokyo, but that is associated with a high cost. With a relaxed delivery constraint of 80ms or more, MultiPub discovers cheaper solutions that use different clouds, until finally, for delivery time bounds of 145ms and above, MultiPub finds a configuration that uses a single European server to serve all clients in Asia. As a result, MultiPub significantly lowers the costs, achieving savings of 36% ($74 / day instead of $120/day). In the South America experiment (figure 5b), the cost savings achieved by MultiPub are even more significant, since outgoing bandwidth costs in this EC2 region are the most expensive. MultiPub can meet a delivery constraint max_T of 60ms for 95% of the publication messages at a very high cost ($210 / day). For values of $max_T \geq 190$, MultiPub determines that an alternate configuration with servers in North America (Virginia) is suitable. In that case, costs for one day descend to $74, representing a saving of 65%.

F. Experiment 4 - Runtime Analysis

Determining the optimal solution is, as mentioned before, exponential with the number of servers. Furthermore, it is linear with the number of messages that have to be considered as we have to sort the messages in order to determine the delivery percentile. In turn, the number of messages increases linear with the number of subscribers on one hand, and, if all publishers send at the same rate, linear with the number of publishers. A first experiment looks at runtimes for a 10-region system when both the number of publishers and subscribers increase to up to 100 (each publisher publishing once every second), showing that at 100 subscribers and 100 publishers it takes a bit less than two minutes to determine the optimal configuration (figure 6a). Similar results are observed when we fix the number of subscribers to 10 and increase the number of publishers to 1000, or when we fix the number of publishers to 10 and increase the number of subscribers to 1000 (graphs omitted due to space constraints). Figure 6b shows the exponential influence of the number of regions by depicting the runtime of the solver for 100 subscribers and publishers with increasing number of regions. With a setting of 5 regions, it took the solver only 3 seconds to find the optimal configuration, that is, around 95% less time than with 10 regions. Which means, that considering only five regions, configurations with 10 publishers and 35,000 subscribers could be determined in less than two minutes.

In summary, these performance measurements confirm that the optimization problem that MultiPub needs to solve can easily be tackled for realistic settings. Already our unopti-

mized, brute-force Python implementation can handle a large number of users, which is largely sufficient to adjust to load changes due to the arrival of new publishers and/or subscribers. To support bigger settings, one could use a more optimized implementation. Also, different topics can be solved in parallel, as they are independent. To support extra-large scale settings, a defendable way to reduce solve time is to only consider a subset of cloud regions, which as we have seen, has an exponential impact on the search space. Simple pruning can remove expensive regions with no or very few subscribers, for instance. Also, proportional bundling can be used, grouping clients that are close to each other and replacing them with a virtual client in order to reduce the scale of the problem.

VI. Related Work

To the best of our knowledge, MultiPub is the first attempt at proposing a pub/sub system that uses an optimization approach considering both delivery times and cloud costs to determine cost-effective and performant deployments in a global-scale setting. Cloud costs have been considered in [24], in which the authors propose a model for efficient resource allocation in topic-based pub/sub systems, in order to reduce cloud-related costs and satisfy subscribers' interest. Similar to them, MultiPub also solves an optimization problem, but our constraints consider message delivery time, and we consider a global-scale deployment of our service in many cloud regions.

PubSubCoord [7] provides a multi-layered, broker-based cloud coordination system for WAN-scale pub/sub systems. IndiQoS [13] proposes QoS guarantees in broker-based publish-subscribe systems. However, it requires a QoS data network, which makes it impractical for Internet and/or cloud deployments. In [18], the authors propose DCRD, a routing algorithm for broker-based topic-based pub/sub systems which selects the most optimal path towards recipients. However, its multi-hop nature can lead to increased delivery times, is not tailored for cloud deployments, and does not take delivery costs into consideration. In [19], the authors propose a scalable pub/sub-based event dissemination platform built on the XMPP protocol. Again, it does not specifically optimize delivery time and does not discuss the issue of optimizing delivery costs. While [23] is not about minimizing costs, the authors nevertheless propose a model that aims at maximizing topic-based pub/sub subscribers' satisfaction in resource-constrained environments.

[10] (survey) discusses the different approaches at providing QoS in the context of wide-scale pub/sub systems. In some systems, an expiration time can be set for publications and all expired publications are rejected by subscribers [6]. The Data Distribution Service (DDS) from the OMG group [4] proposes the specification of a rich software architecture for QoS-constrained data delivery schemes, with an emphasis on pub/sub applications. However, it is not tailored for the cloud and it does not provide or specify an implementation, hence performance depends on specific standard implementations.

Besides topic-based pub/sub, content-based is another well-known pub/sub flavor [22], where subscriptions are done over predicates computed on the publications themselves, which can offer more flexibility at the expense of higher computation costs. Some cloud content-based pub/sub services have been proposed such as [21], [8]. Graph-based pub/sub is yet a different paradigm in which interests are represented as graph subscriptions [11]. We also mention that some entreprises also provide large-scale pub/sub services in the cloud [5], [2], [1].

VII. Conclusion

We presented MultiPub, a cost-minimizing topic-based pub/sub cloud middleware for latency-constrained applications with clients that are potentially distributed all over the globe that require strict delivery time bounds. As future work, we plan on proposing heuristic-based clustering approaches to support even larger-scale systems. We also plan to extend our model to support content-based pub/sub systems. Since these systems need more CPU resources, such an extension would also need to take VM usage / CPU costs into consideration.

References

[1] Amazon SNS. *http://aws.amazon.com/fr/sns/* (2014)
[2] Google Cloud Messaging. *https://developer.android.com/google/gcm/index.html* (2014)
[3] Amazon Elastic Compute Cloud. *https://aws.amazon.com/ec2/* (2016)
[4] Data Distribution Service 1.4. *http://www.omg.org/spec/DDS/1.4/* (2016)
[5] Google Cloud Pub/Sub. *https://cloud.google.com/pubsub/* (2016)
[6] Java Message Service Specifications v1.1 (2016)
[7] An, K., Gokhale, A., Tambe, S., Kuroda, T.: Wide area network-scale discovery and data dissemination in data-centric publish/subscribe systems. pp. 6:1–6:2. Middleware Posters and Demos '15, ACM (2015)
[8] Barazzutti, R., Heinze, T., Martin, A., Onica, E., Felber, P., Fetzer, C., Jerzak, Z., Pasin, M., Riviere, E.: Elastic scaling of a high-throughput content-based publish/subscribe engine. In: ICDCS. pp. 567–576 (2014)
[9] Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., Claypool, M.: The effects of loss and latency on user performance in unreal tournament 2003®. pp. 144–151. NetGames '04, ACM (2004)
[10] Bellavista, P., Corradi, A., Reale, A.: Quality of service in wide scale publish-subscribe systems. IEEE Communications Surveys Tutorials 16(3), 1591–1616 (Third 2014)
[11] Cañas, C., Pacheco, E., Kemme, B., Kienzle, J., Jacobsen, H.A.: Graps: A graph publish/subscribe middleware. pp. 1–12. Middleware '15, ACM
[12] Cañas, C., Zhang, K., Kemme, B., Kienzle, J., Jacobsen, H.A.: Publish/subscribe network designs for multiplayer games. In: Middleware 2014. pp. 241–252 (2014)
[13] Carvalho, N., Araujo, F., Rodrigues, L.: Scalable qos-based event routing in publish-subscribe systems. In: Fourth IEEE International Symposium on Network Computing and Applications. pp. 101–108 (July 2005)
[14] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. 35(2), 114–131 (2003)
[15] Gascon-Samson, J., Garcia, F.P., Kemme, B., Kienzle, J.: Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. In: ICDCS 2015. pp. 486–496 (June 2015)
[16] Gascon-Samson, J., Kienzle, J., Kemme, B.: Dynfilter: Limiting bandwidth of online games using adaptive pub/sub message filtering. In: Network and Systems Support for Games (NetGames), 2015 International Workshop on. pp. 1–6 (Dec 2015)
[17] Gummadi, K.P., Saroiu, S., Gribble, S.D.: King: Estimating latency between arbitrary internet end hosts. In: ACM SIGCOMM Workshop on Internet Measurment (IMW). pp. 5–18 (2002)
[18] Guo, S., Karenos, K., Kim, M., Lei, H., Reason, J.: Delay-cognizant reliable delivery for publish/subscribe overlay networks. In: ICDCS 2011. pp. 403–412 (June 2011)
[19] Hong, R., Shin, S., Yoon, Y., Laxmankatole, A., Woo, H.: Global-scale event dissemination on mobile social channeling platform. In: MobileCloud 2014. pp. 210–219 (April 2014)
[20] Kienzle, J., Verbrugge, C., Kemme, B., Denault, A., Hawker, M.: Mammoth: a massively multiplayer game research framework. In: Foundations of Digital Games (FDG). pp. 308–315 (2009)
[21] Li, M., Ye, F., Kim, M., Chen, H., Lei, H.: A scalable and elastic publish/subscribe service. In: IPDPS. pp. 1254–1265 (2011)
[22] Rosenblum, D.S., Wolf, A.L.: A design framework for internet-scale event observation and notification. In: ESEC. pp. 344–360 (1997)
[23] Setty, V., Kreitz, G., Urdaneta, G., Vitenberg, R., van Steen, M.: Maximizing the number of satisfied subscribers in pub/sub systems under capacity constraints. In: IEEE INFOCOM 2014. pp. 2580–2588
[24] Setty, V., Vitenberg, R., Kreitz, G., Urdaneta, G., van Steen, M.: Cost-effective resource allocation for deploying pub/sub on cloud. In: ICDCS 2014. pp. 555–566