Resource Management for Elastic Publish Subscribe Systems: A Performance Modeling-based Approach

Phuong Nguyen
Department of Computer Science
University of Illinois at Urbana-Champaign
pvnguye2@illinois.edu

Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
klara@illinois.edu

***Abstract*—As more and more information systems are moving to the cloud, there have been efforts to deploy publish-subscribe (or pub/sub) systems in the cloud environment to take advantage of the elasticity of resources. As a result, there is a need to perform resource management for the cloud-based pub/sub systems that support various types of jobs, each consists of a series of tasks, or a workflow. Designing an efficient and effective resource management approach for the cloud-based pub/sub system is challenging because such an approach needs to be able to support flexible provisioning strategies, model the complex interactions between heterogeneous types of jobs, and provide dynamic resource allocation capability. In this paper, we formulate the resource management problem of elastic pub/sub system as optimization problems using different objectives functions. We model the elastic pub/sub system as a multiple-class open queuing network to derive system performance measures, and propose greedy algorithms to efficiently solve the optimization problems. Our evaluation based on simulation on real system show that our proposed solution outperforms the baseline and is robust in dealing with high-volume and fast-changing workload.**

I. Introduction

Publish-subscribe system (or pub/sub system for short) [12] is a commonly used asynchronous communication pattern for a variety of applications. In such the system, the asynchronization is implemented by decoupling the producers and consumers of messages[1]. Particularly, messages generated by producers will be sent to the pub/sub system and are kept under different message queues. A consumer registers its interest to certain type of messages and the pub/sub system will deliver messages to consumers of appropriate message types.

There are two main types of pub/sub system based on how the subscription works: topic-based and content-based. Topic-based is the simplest form of pub/sub system where the messages are associated with topic strings and subscriptions are defined at the level of topics. Content-based pub/sub allows more complicated subscriptions based on the content of the message. However, as complexity often comes at the trade-off for efficiency, topic-based pub/sub is thus much more efficient than content-based one, due to its simple message-to-consumer matching. As a result, topic-based pub/sub is more popular in real-world applications and is the type of system we study in this paper.

The asynchronous message passing mechanism in pub/sub system gives applications the flexibility to decide the logic of how to react to events and what chain of the steps needed to process an event. As a result, pub/sub systems are often used to process complex jobs that involve a series of tasks in form of a Directed Acyclic Graph (DAG), or workflow[2]. Examples include systems that support execution of scientific computing workflows [30][31], business processes [32], and increasingly popular Internet of Things applications [33]. In such systems, the topics, each represents a message queue for a type of task, are loosely coupled with each other via message passing (i.e., as a task is finished, it places a message(s) for the subsequent task(s) into the appropriate message queue(s)).

With the wide range of applications supported by pub/sub system and the increasing popularity of cloud infrastructure, there have been efforts [2][3][6] to deploy pub/sub system in the cloud environment to take advantage of the elasticity of the cloud. Particularly, as data generated by producers increases, additional cloud resources can be added to the pub/sub system dynamically to make it scalable. One scaling strategy is to increase the capacity of message queues (so that more produced messages can be stored until they are consumed). Another strategy is to increase the number of consumers per topic, so that more messages can be processed in parallel. Given that processing messages is the main bottleneck and messages are often small in size (i.e., messages only store job description, not actual data), elastic scaling of consumers is the more desirable approach and is the one studied in this paper.

Designing an efficient resource management approach for the cloud-based pub/sub system is challenging due to a number of reasons. First, similar to other cloud deployment, it is desirable that the cloud-based pub/sub system should be able to support different provisioning strategies to satisfy different objectives set by users, such as quality of service

[1]We use publisher/producer and subscriber/consumer interchangeably.

[2]We use job and workflow interchangeably, both refer a DAG of tasks.

(e.g., response time, utilization), or cost of resources. Second, the heterogeneity of input jobs, each job is in form of a DAG of tasks, makes it difficult to model complex interactions between topics in pub/sub system (where each topic corresponds to a type of task). Third, since real-world applications often have variable and sometimes bursty loads, static resource allocation and rule-based provisioning strategies are not suitable.

In this paper, in order to tackle the above issues, we propose a novel system architecture for cloud-based pub/sub system that supports execution of heterogeneous workflows and a performance modeling-based approach for resource management of elastic pub/sub system. Our proposed architecture separates *control plane*, which manages resources and all the execution logic of workflows, and *compute plane*, which focuses on actual processing of workflow's tasks, and thus, allows scalable and flexible implementation of heterogeneous workflows/jobs. To support *flexible provisioning*, we formulate the resource management problem as optimization problems with different objectives (i.e., minimizing response time or minimizing cost of resources). To model the complex interactions between topics to support *heterogeneous jobs*, we propose to model the performance of the system as a multiple-class open queuing network. From this modeling, based on different assumptions about the distributions of arrival rates of jobs and processing rates of tasks, we are able to obtain (analytical or approximate) solutions for system performance measures. To support *dynamic resource provisioning*, we propose greedy strategies to efficiently solve the optimization problems. Our evaluation results show that our proposed solutions outperform the baseline and are robust in dealing with high-volume and fast-changing workload.

The paper is organized as follow. In Section II, we describe in details the proposed system architecture of cloud-based pub/sub system that supports execution of heterogeneous workflows. After that, in Section III, we formally define the resource management problem as optimization problems. In Section IV, we describe our performance modeling of elastic pub/sub system using generalized multiple-class open queuing network. We propose greedy strategies to efficiently solve the optimization problems in Section V. In Section VI, we the evaluation results on the performance of the proposed approach. We summarize some related work in Section VII. Finally, we conclude the paper and discuss some future directions in Section VIII.

II. Cloud-based Publish Subscribe System

Our proposed system architecture for cloud-based pub/sub system that support execution of heterogeneous workflows is presented in Figure 1. The system consists of three main components: front-end, control plane, and compute plane. The system can be deployed on any private cloud or public cloud's IaaS offering.

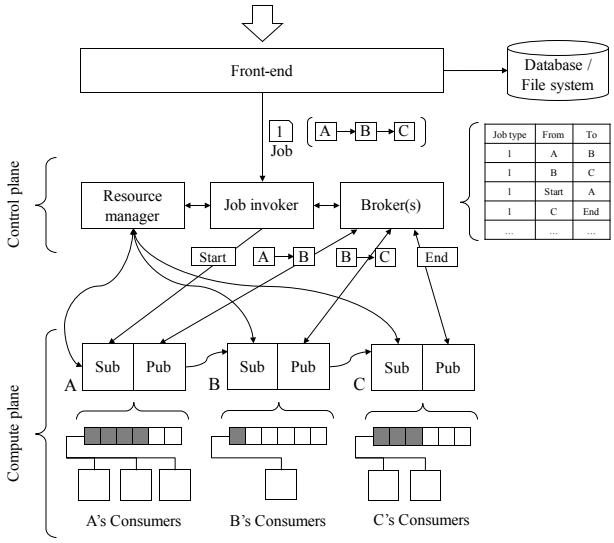


Figure 1: Cloud-based pub/sub system architecture.

Front-end is the entry point for all incoming requests. Any input data that come with the requests are stored into a database or file system (which will then be accessible when requests are processed). Front-end translates the incoming request into a *job profile* that includes time-stamp, job ID, job type, and any references to its input data stored in database/file system.

Control plane manages resources and handles all the execution logic of jobs. When the *Job invoker* receives the job profile from front-end, it will ask its *brokers* which task of the job should be processed first. The brokers maintain a mapping table that includes all the task dependencies of the job types that system supports. Particularly, given a job type and a current task (i.e., "From" field), the brokers will return what is the next task (i.e., "To" field) to be processed for a job. The Job invoker forward the job profile to the appropriate component in compute plane where the first task of the job will be processed.

Compute plane is in charge of actual processing of tasks of a job. It consists of a "peer-to-peer"-like network of processing components, which are commonly abstracted as *topics* in pub/sub system, each is responsible for processing a particular type of task. Each topic operates both as a subscriber and a publisher. As a *subscriber*, a topic maintains a *message queue* for requests of the task type it is in charge for and a set of *consumers* that subscribes to the queue to process the requests (the number of consumers per topic can be adjusted dynamically and is the topic of study for resource management). Each consumer of a topic also acts as a *publisher*. After a task is processed, the consumer will ask control plane's brokers for the subsequent task(s) of the job and then, forward the job request to the appropriate topic(s).

In the Figure 1's example, the front-end sends a request of a type-1 job (which consists of three tasks $A \rightarrow B \rightarrow C$ that

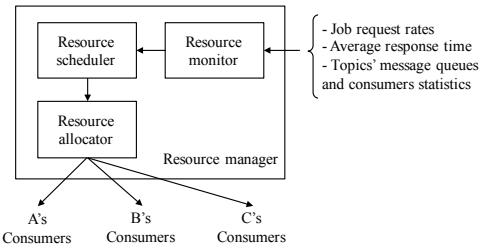


Figure 2: Resource manager's components.

are executed consecutively) to the control plane. Job invoker asks the brokers what is the first task for a job of type 1 (i.e., task A) and then forwards the job request to the appropriate topic in charge of task A. The job request is pushed into the message queue and is processed by one of the consumers of topic A. After finishing processing the job request, topic A's consumer asks brokers which topic(s) should it sends the job request to next. As the brokers return the next destination as the topic B, the consumer will publish the job request to the message queue of topic B. Similar procedure applies for the transition from task B to C. The processing of the job request ends when task C's consumer is notified that task C is the last task of a type-1 job.

Resource manager, which is part of control plane, consists of three components: monitor, scheduler, and allocator (Figure 2). *Resource monitor* collects various statistics of the system in real-time, such as job request arrival rates, actual job response time, and feeds these information to scheduler. *Resource scheduler* implements resource management algorithms and decides whether to perform rescheduling of resources based on monitoring information (e.g., when system's average response time is greater than a certain predefined threshold). If a rescheduling is needed, resource scheduler execute appropriate algorithm and produce a new allocation of resources over topics (i.e., how many consumers are needed for each topic). The rescheduling decision is sent to *resource allocator* that actually performs provisioning of resources over topics.

How to design efficient and effective resource management algorithms for elastic pub/sub system is the main focus of this paper.

III. Resource Management Problem Definition

Let us consider a cloud-based pub/sub system that consists of J topics (i.e., supports processing J tasks) and accepts requests for N types of jobs, each job corresponds to a workflow of tasks supported by the pub/sub system. For each type of job i, we assume that the arrival rate of requests follows a general distribution, denoted by expected rate λ_i and squared coefficient of variation (or *scv* for short) of the rate ca_i^2. The set of parameters $\Lambda = \{(\lambda_i, ca_i^2)\}(1 \leq i \leq N)$ defines the system's workload.

In terms of computational parameters, for each topic j $(1 \leq j \leq J)$, there are m_j (uniform) consumers subscribe to its message queue. For each consumer of a topic j, we assume that the time it takes to process the appropriate task follows a general distribution, denoted by expected processing rate μ_j and scv of the rate cs_j^2. We assume that the processing rate parameters $\Gamma = \{(\mu_j, cs_j^2)\}(1 \leq j \leq J)$ depend on the implementation of consumers and task input data, and are given (e.g., by the collecting statistics of the processing time of completed tasks).

Since the workload and computational times could be considered as given, the numbers of consumers over topics $\mathbf{m} = (m_1, m_2, ..., m_J)$ (which can be dynamically provisioned by exploiting the elasticity of the cloud infrastructure) are the main variables to measure performance of the elastic pub/sub system.

The system performance metrics include job's expected response time and cost of computational resources. Since response time is linearly related to the number of job requests being in the system (by Little's law), we use *work-in-progress*, denoted as WIP, as the performance metric for time. Particularly, WIP of the system is defined as $WIP(\mathbf{m}) = \sum_{j=1}^{J} \nu_j L_j(m_j)$, with ν_j and $L_j(m_j)$ are respectively the value of a job request (assumed to be given) and the number of job requests in progress at topic j (if $\nu_j = 1, \forall j$, WIP equals the total number of jobs in the system). In terms of the resource cost, since in this paper we consider allocating consumers over topics is the main resource allocation mechanism, the total resource cost depends on the number of provisioned consumers per topic and is define as $F(\mathbf{m}) = \sum_{j=1}^{J} F_j(m_j)$, where the function $F_j(m_j)$ (assumed to be given) is the cost of allocating m_j servers at station j. As we will show in Section IV, since the performance of resource allocation algorithms depends on the shape of $F_j(m_j)$, we assume that $F_j(m_j)(\forall 1 \leq j \leq J)$ need to be a *non-decreasing convex function* of m_j. This assumption is reasonable since the resource cost increases as the number of consumers at a topic increases.

With the above notations and definitions, the resource management problem for cloud-based pub/sub system can be formulated as optimization problems. By using different objective functions for optimization problems, we allow users to flexibly choose between different resource provisioning strategies to suit their purposes.

For the first optimization problem, the objective is to minimize system's overall response time, or appropriately the WIP metric:

Problem Definition 1: (Minimal Time Resource Allocation) Given a cloud-based pub/sub system that supports N types of job and J different tasks, a workload Λ, processing rates Γ, and a cost budget $\mathcal{M}$, find an optimal allocation $\mathbf{m}$ of consumers to topics to minimize system's work-in-progress WIP:

$$\underset{\mathbf{m}}{\text{argmin}} \quad WIP(\mathbf{m}) = \sum_{j=1}^{J} \nu_j L_j(m_j)$$
$$\text{subject to} \quad \sum_{j=1}^{J} F_j(m_j) = \mathcal{M}$$

For the second optimization problem, the objective is to minimize the total resource cost of allocating consumers across topics:

Problem Definition 2: (Minimal Cost Resource Allocation) Given a cloud-based pub/sub system that supports N types of job and J different tasks, a workload Λ, processing rates Γ, and a WIP (or time) constraint $\mathcal{T}$, find an optimal allocation $\mathbf{m}$ of consumers to topics to minimize system's total resource cost $F(\mathbf{m})$:

$$\underset{\mathbf{m}}{\text{argmin}} \quad F(\mathbf{m}) = \sum_{j=1}^{J} F_j(m_j)$$
$$\text{subject to} \quad \sum_{j=1}^{J} \nu_j L_j(m_j) \leq \mathcal{T}$$

In order to solve the above problems, it is important to obtain the formulation for the performance metric WIP. In the next section, we will describe our proposed approach to derive WIP of the elastic pub/sub system using queuing theory.

IV. Modeling Performance of Elastic Pub/Sub System

A. Modeling Motivation

From the system architecture description in Section II, it is intuitive to model each topic as a *queue* (i.e., represented by the topic's message queue) with multiple *workers* (i.e., the subscribing consumers). In addition, topics in the system are connected to each other because job requests are routed through the topics following the dependencies between tasks in a job. Hence, we can model the system as a *network of queues*, with each topic is an individual queue. Besides, as job requests can be of different job types and they arrive then leave the system as they are finished, the queuing network model of the system is *multi-class* and *open*.

By modeling the elastic pub/sub system as a multiple-class open queuing network (OQN), we are able to apply known results in queuing theory [15][18][17] to obtain the solution for the system's performance metrics. However, since there are numerous models have been developed for OQN, choosing an appropriate one is non-trivial. While other related work that utilize queuing network for performance modeling often opt for simplified models to obtain analytical solutions, they are limited by strong (and sometimes unrealistic) assumptions about the system, such as deterministic or exponential distribution of arrival rates of job requests and processing rates.

In this paper, we decide to build our model based on more realistic assumptions. Particularly, we consider job request arrival rates and processing time at each topic both follow general distributions, represented by parameter sets $\Lambda = \{(\lambda_i, ca_i^2)\}(1 \leq i \leq N)$ and $\Gamma = \{(\mu_j, cs_j^2)\}(1 \leq j \leq J)$, respectively. Under these assumptions, each topic is appropriate to a *GI/G/m* queue and the elastic pub/sub system can be modeled as a *Generalized Multiple-class Jackson OQN* [15][18]. In the remaining of this section, we show how to leverage this model to obtain solution for performance metric of the system (i.e., WIP).

B. From Model to Performance Characterization

Before analyzing our proposed modeling using generalized multiple-class Jackson OQN, let us consider the special case, when job arrival rates and task processing rates are exponentially distributed (i.e., $ca_i^2 = 1, \forall 1 \leq i \leq N$ and $cs_j^2 = 1, \forall 1 \leq j \leq J$). In this case, each topic in the pub/sub system is appropriate to a *M/M/m* queue. Because of the exponential distributions, we can aggregate all job types as a single type (since the combination of exponential distribution is also exponential). In addition, we can obtain the analytical solution of the expected number of work-in-progress job requests of a topic j (i.e., L_j) as a function of $\mu_j, \tilde{\lambda}_j, m_j$ following Erlang-C formula [15] (where $\tilde{\lambda}_j$ is the aggregated job arrival rate at topic j of all job types: $\tilde{\lambda}_j = \sum_{i=1}^{N} \lambda_{ij}$ with λ_{ij} is the expected arrival rate of job request type-i at topic j):

$$L_j^{M/M/m}(\mu_j, \tilde{\lambda}_j, m_j) = \frac{\frac{\tilde{\lambda}_j}{\mu_j m_j}(\frac{\tilde{\lambda}_j}{\mu_j})^{m_j}\pi(0)}{(1 - \frac{\tilde{\lambda}_j}{\mu_j m_j})^2 m_j!} + \frac{\tilde{\lambda}_j}{\mu_j} \quad (1)$$

with:

$$\pi(0) = \{ \sum_{t=0}^{m_j - 1} \frac{(\frac{\tilde{\lambda}_j}{\mu_j})^t}{t!} + \frac{(\frac{\tilde{\lambda}_j}{\mu_j})^{m_j}}{(1 - \frac{\tilde{\lambda}_j}{\mu_j m_j}) m_j!} \}^{-1}$$

For generalized case, since the job arrival rates and task processing rates are generally distributed, it is not possible to obtain exact analysis of L_j as in the special case. Hence, in this paper, we employ an approximation method, named *parametric decomposition* [18], to measure the steady-state behavior solution for L_j[3]. Specifically, for each topic j (in general case, is modeled as a GI/G/m queue), the expected number of work-in-progress job requests $L_j^{GI/G/m}$ is derived as a function of $\tilde{\lambda}_j, \tilde{ca}_j^2, \mu_j, cs_j^2$, and m_j (where $\tilde{\lambda}_j, \tilde{ca}_j^2$ are aggregated job arrive rate and scv of all job types at topic j, obtained from the parametric decomposition procedure [18]). Among several good two-moment approximations of $L_j^{GI/G/m}$ that have been established for the GI/G/m queue [34], in this paper, we use the common approximation

[3]Due to space limitation, we leave the detailed formulation in the extended version of this paper [8].

formulation proposed in [35] that is based on an extension of the exact formula used in the M/M/m case:

$$L_j^{GI/G/m}(\tilde{\lambda}_j, \tilde{ca}_j^2, \mu_j, cs_j^2, m_j) = \frac{\tilde{\lambda}_j}{\mu_j} + \tilde{\lambda}_j(\frac{\tilde{ca}_j^2 + cs_j^2}{2})(L_j^{M/M/m}(\mu_j, \tilde{\lambda}_j, m_j) - \frac{\tilde{\lambda}_j}{\mu_j}) \quad (2)$$

where $L_j^{M/M/m}(\mu_j, \tilde{\lambda}_j, m_j)$ is the expected number of job requests in progress of a M/M/m queue as computed in Equation 1.

In Equation 2, we can consider $\tilde{\lambda}_j, \tilde{ca}_j^2, \mu_j, cs_j^2$ as given (i.e., either provided or calculated by parametric decomposition). Therefore, $L_j^{GI/G/m}$ becomes a function of m_j only: $L_j^{GI/G/m}(m_j)$.

Given the performance measure of individual topic $L_j^{GI/G/m}(m_j)$ obtained by Equation 2, the system performance measure (i.e., WIP of the whole pub/sub system) can be calculated as $WIP(\mathbf{m}) = \sum_{j=1}^{J} \nu_j L_j^{GI/G/m}(m_j)$, with ν_j is the value of a job request at topic j. In the following, without any confusion, we use $L_j(m_j)$ to refer to $L_j^{GI/G/m}(m_j)$ for being concise.

V. Greedy Resource Allocation Solutions

With the formulation of system's performance metric WIP obtained from previous section, we now show how to efficiently solve the optimization problems described in Section III.

While we can view both optimization problems in Definition 1 and 2 as integer programming problems and apply standard solver to solve them, dynamic resource allocation for the system require more efficient solutions. In this paper, we propose greedy strategies to efficiently solve the optimization problems. In addition, by realizing the convex property of the objective functions, we are able to prove that the solutions by greedy algorithms are also the optimal solutions.

For the first optimization problem (Definition 1), by observing that $L_j(m_j)$ is a convex non-increasing function of m_j, $\forall 1 \leq j \leq J$ [19], we can solve the optimization problem in Definition 1 using a greedy strategy. Particularly, in Algorithm 1, each topic is initialized with one consumer, and then, the algorithm greedily finds the topic with the largest *benefit* if being allocated one more consumer. The benefit is defined to be proportional to the decrease of the number of work-in-progress job requests (i.e., $\nu_j[L_j(m_j^{i-1}) - L_j(m_j^{i-1} + 1)]$). The algorithm ends when it reaches the resources cost constraint $\mathcal{M}$.

With the non-increasing convexity of $L_j(m_j)$, it can be proven that the solution of Algorithm 1 is also the optimal solution, based on Theorem 3 in [17].

For the second optimization problem (Definition 2), given the non-decreasing convexity of $F_j(m_j)$, $\forall 1 \leq j \leq J$ (as assumed) and the non-increasing convexity of $L_j(m_j)$, $\forall 1 \leq j \leq J$, we can again use the similar greedy strategy as in Algorithm 1 to find the optimal resource allocation solution. The minimal cost greedy resource allocation algorithm is presented in Algorithm 2.

Algorithm 1 Minimal Time Greedy Resource Allocation

1: **procedure** MinTimeGreedy
2: Initial allocation $\mathbf{m}^0$: $m_j^0 = 1, \forall 1 \leq j \leq J$
3: $i = 1$ ▷ Initialize iteration count
4: **while** $\sum_{j=1}^{J} F_j(m_j) < \mathcal{M}$ **do**
5: Find $j* = argmax_{1 \leq j \leq J} \nu_j[L_j(m_j^{i-1}) - L_j(m_j^{i-1} + 1)]$
6: $m_{j*}^i = m_{j*}^{i-1} + 1$ ▷ Add one consumer to most benefit topic
7: $i = i + 1$
8: Return $\mathbf{m}^i$

Algorithm 2 Minimal Cost Greedy Resource Allocation

1: **procedure** MinCostGreedy
2: Initial allocation $\mathbf{m}^0$: $m_j^0 = 1, \forall 1 \leq j \leq J$
3: $i = 1$ ▷ Initialize iteration count
4: **while** $WIP(\mathbf{m}^i) \leq \mathcal{T}$ **do**
5: Find $j* = argmax_{1 \leq j \leq J} \frac{\nu_j[L_j(m_j^{i-1}) - L_j(m_j^{i-1}+1)]}{F_j(m_j^{i-1}+1) - F_j(m_j^{i-1})}$
6: $m_{j*}^i = m_{j*}^{i-1} + 1$ ▷ Add one consumer to most benefit topic
7: $i = i + 1$
8: Return $\mathbf{m}^i$

The main difference between Algorithm 2 and 1 is that, in Algorithm 2, the benefit of adding an additional consumer to a topic is defined to be inversely proportional to the increase in resource cost (i.e., $F_j(m_j^{i-1}+1) - F_j(m_j^{i-1})$) and directly proportional to the decrease of the number of work-in-progress job requests (i.e., $\nu_j[L_j(m_j^{i-1}) - L_j(m_j^{i-1}+1)]$) (Line 5). Based on Theorem 2 in [17], the solution by Algorithm 2 is proven to be "sufficiently close to the optimal solution".

VI. Evaluation

In this section, we evaluate the effectiveness of our proposed approach compared to baseline in the two resource management tasks defined in Section III.

A. Evaluation Settings

Implementation: We implemented the proposed cloud-based elastic pub/sub system using RabbitMQ[4] as the message queue engine and Docker[5] container technology as the implementation platform for consumers (for better isolation and server consolidation). Particularly, each consumer is implemented and encapsulated into a Docker image and subscribes to a RabbitMQ's message queue of appropriate topic. We deployed the system on a cluster of three servers, each server is equipped with an Intel Xeon quad core processor (1.2Ghz for each core) and 16GB of RAM. We use Kubernetes[6] as the Docker container ostrastration

[4]RabbitMQ - https://www.rabbitmq.com
[5]Docker - https://www.docker.com
[6]Kubernetes - http://kubernetes.io

Task	Description	μ_j	cs^2_j
A	Unpacking digital microscope output files (e.g., DM3, HDF5)	4.2	0.33
B	Extracting and analyzing metadata from input file	3.7	0.5
C	Extracting and analyzing image from input file	6.7	0.4
D	Classify the input file into appropriate experiment type and predict if the experiment is successful or not	5.1	0.5

(a) Supported types of task.

Job type	Format	ca^2_j
1	A → B → D	0.33
2	A → B, A → C, B → D, C → D	0.5
3	C → D	0.25

(b) Supported types of job.

Figure 3: Tasks and jobs supported by the system.

engine for the cluster and each topic's consumer set is abstracted as a Kuberneters' ReplicationController. The resource manager (resource allocator in particular) interacts directly with Kubernetes to dynamically scale the size of ReplicationController (i.e., number of consumers) of each topic. All system components are implemented using Python programming language.

Case study: We take the application of executing scientific computing workflows as the case study. Particularly, the system supports analyzing experimental data generated by digital microscopes (which are usually in forms of DM3, or HDF5 files). Four types of task are supported, which correspond to the steps needed to process input data (Figure 3a). Depending on the input data, the system can support three different types of job, each job consists of all or a subset of supported tasks (Figure 3b).

Parameter settings: The processing rates of tasks are given in Figure 3a. The scv of job arrival rates are given in Figure 3b, while the expected arrival rates of each job type (i.e., λ_i) are varied during the evaluation to represent changing workload [7]. Please note that the time unit we use for rates (i.e., processing time rate μ_j and job arrival rate λ_i) is *per minute*. To simplify the computation, we use a uniform resource cost function, i.e., $F_j(m_j) = m_j, \forall j$, and consider the job requests as equally important, i.e., $\nu_j = 1, \forall j$[8].

In terms of comparing approach, we compare our resource management algorithms, named `MinTime` (Algorithm 1) and `MinCost` (Algorithm 2), with random resource allocation approach, named `Random`. In `Random`, for each iteration, a topic is randomly chosen to be allocated an additional consumer. To evaluate the performance of different algorithms, we initially allocate one consumer to each topic: $\mathbf{m} = (1, 1, 1, 1)$. Then, after each iteration (i.e., after a consumer is allocated to a topic), we measure the average response time of each type of job, as well as the average of all jobs. An algorithm is considered better if it achieves lower average response time after a given number of iterations (in case of minimal time allocation), or requires less iterations to reach a predefined response time threshold (in case of minimal cost allocation).

[7]Although these parameters are assumed to be given, they can be automatically captured by using job profiling techniques or by collecting statistics of the submitted jobs overtime.

[8]Please note that $F_j(m_j)$ and ν_j can be chosen in any form so that $WIP(\mathbf{m})$ and $F(\mathbf{m})$ maintain their non-increasing and non-decreasing convex properties.

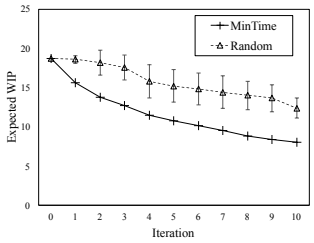


Figure 4: Numerical analysis comparison.

B. Numerical Analysis

First, we compare our proposed algorithm, `MinTime` in particular, with `Random` using numerical analysis. Specifically, given a workload $\{\lambda_i\} = (3.0, 3.5, 3.0)$ and a cost constraint $\mathcal{M} = 10$ (since $F_j(m_j) = m_j$, $\mathcal{M}$ is equivalent to the number of additional consumers allowed), we calculate the number of expected work-in-progress jobs in the system (i.e., $WIP(\mathbf{m})$ as derived in Section IV) produced by each algorithm after each iteration (i.e., an iteration is equivalent to an additional consumer added). The result in Figure 4 shows that `MinTime` outperforms `Random` by adding consumers to the most benefit topics, and thus helps reduce $WIP(\mathbf{m})$ at a faster rate. We observe similar result when comparing `MinCost` with `Random`.

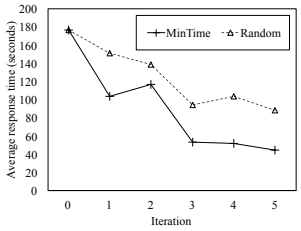
We also notice that the result of `Random` can be different between different runs of `Random` algorithm (hence the error bars). Therefore, in the remaining of this section, we will use the `Random`'s best result after multiple runs.

C. Minimal Time Optimization Task

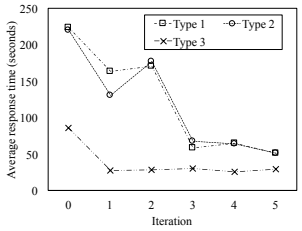
For the Minimal Time Resource Allocation task, given a workload $\{\lambda_i\} = (2.0, 2.5, 2.0)$ and a cost constraint $\mathcal{M} = 5$, we perform resource allocation using `MinTime` and `Random`. We measure the performance of each algorithm over iterations. Figure 5a shows that, as two algorithms reach the cost constraint (i.e., after 5 iterations), `MinTime` outperforms `Random` by achieving a lower average response time of all types of job. Although `Random`'s allocation helps reduce the response time at some degree, it could not achieve optimal result due to its randomization in selecting topics for provisioning. In addition, `MinTime` also performs well with individual types of job. Figure 5b shows that the average response time of each type of job quickly drop after just a few consumers are added to the system.

D. Minimal Cost Optimization Task

For the Minimal Cost Resource Allocation task, given a workload $\{\lambda_i\} = (3.0, 3.5, 3.0)$ and a response time



(a) Average response time over all types of job.



(b) `MinTime`'s performance with different types of job.

Figure 5: Minimize time resource allocation comparison.

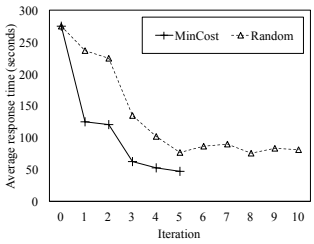


Figure 6: Minimize cost resource allocation comparison.

constraint of 50 seconds: $\mathcal{T} = 50$, we perform resource allocation using `MinCost` and `Random` until the system average response time of all types of job smaller than or equal $\mathcal{T}$. The result in Figure 6 shows that `MinCost` satisfies the response time constraint in just 5 iterations (i.e., 5 additional consumers are needed). On the other hand, even though `Random` helps reduces the response time, it struggles in bringing down the response time to below $\mathcal{T}$, even after 10 iterations.

The results in both optimization tasks help confirm the effectiveness of using greedy strategy in selecting the topics for resource provisioning that maximize the overall benefit.

E. Efficient Dynamic Provisioning

In this section, we evaluate the efficiency of our proposed resource management solution when dealing with changing workload. Particularly, we create a bursty workload that consists of 100 job request for each type of job. The first 20% of the requests arrive with rates $\{\lambda_i\} = (0.5, 1.0, 0.5)$ and the remaining 80% of the requests (abnormally) arrive with rates multiple times higher $\{\lambda_i\} = (3.0, 3.5, 3.0)$. At the beginning of the test, each topic has one consumer: $\mathbf{m} = (1, 1, 1, 1)$. Our resource manager is configured to run during the test using `MinTime` algorithm and cost constraint $\mathcal{M} = 5$.

The response time statistics of all requests during the test period is shown in Figure 7. We can see that the resource manager observes the increase in the average response time of the system (thanks to the resource monitor) and quickly comes up with a rescheduling strategy of the resources to bring the average response time back to the level before bursty load happens. Specifically, the resource scheduler leverages all allowed resource cost and generates a new allocation of resources over topics: $\mathbf{m} = (2, 2, 1, 4)$. This new allocation decision is executed by resource allocator using Kubernetes' container scaling capability. The whole process from observing the increasing response time, generating new rescheduling strategy, to re-scaling the system is efficient, and thus, the bursty load only affects a small portion of requests (about 15% of requests) during a short amount of time.

Figure 7: Dynamic provisioning to deal with bursty workload.

VII. Related Work

Publish-subscribe system [21][12], with its wide range of applications, has been a large topic of study. The related work can be categorized based on the types of pub/sub system: topic-based [22] and content-based [23]. In addition, the pub/sub systems can also be peer-to-peer [24][25] or cloud-based [2]. In this paper, we focus on topic-based pub/sub systems that are deployed on the cloud.

There have been a lot of efforts recently [2][3][6] to deploy pub/sub system in the cloud environment to take advantage of the elasticity of the cloud. For example, Li et al. [2] exploit the skewness of workload to achieve high performance content-based pub/sub system. Gascon et al [3] propose a cloud resource provisioning strategy for pub/sub system based on monitoring the incoming workload. Setty et al. [4] study the resource cost-effective deployment problem of pub/sub system with known workload. This paper is the first one that derives a performance models for elastic pub/sub system that supports multiple types of jobs and allows flexible provisioning strategies.

Most of the efforts on resource management for cloud-based systems have been on batch processing [26] or interactive big data analytics systems. Fu et al. [5] model the performance of the cloud-based data stream processing system that supports one type of job in synchronous scenario. In this paper, we focus on resource management for real-time asynchronous pub/sub system that can support multiple types of jobs. Our proposed resource allocation strategies can be used with other more generic cloud resource management solutions, such as YARN [29] and Mesos [28] that help allocate available computational resources to applications.

Related work on elasticity controller [7][10][11] and resource provisioning of multi-tier applications [14][9][16]

have focused on designing autoscaling mechanism for elastic cloud system. However, the main difference between our work and these work is that they often model the cloud system as a single job shop system that supports one type of application, or multiple independent applications. Whereas, our proposed system needs to handle heterogeneous types of jobs that consist of interconnected tasks (or applications). Also, instead of making ad-hoc provisioning decisions, we model the resource allocation problem for cloud-based pub/sub system as the optimization problems and solve them using efficient greedy algorithms with guaranteed results. In addition, our use of containers as the abstraction of resources, instead of VMs or servers as in related work, offers finer-grained elasticity scaling for applications.

VIII. Conclusions and Future Work

In this paper, we formulate the resource management problem of elastic pub/sub system as optimization problems using different objectives functions. We model the elastic pub/sub system as a multiple-class open queuing network to derive system performance measures, and propose greedy algorithms to efficiently solve the optimization problems.

For the future work, we would like to study the effect of routing optimization to the performance of the system when multiple possible paths exist for a given class (i.e., when the task dependencies are not strictly required). We would like to investigate to incorporate the result by Kameda et al. [20] that shows the uniqueness of the solution for optimal static routing in open BCMP queuing networks.

Acknowledgment

This research was funded by the National Science Foundation NSF ACI 1443013. The opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of the National Science Foundation.

References

[1] Oh, S., Kim, J. H., and Fox, G.. *Real-time performance analysis for publish/subscribe systems*. Future Generation Computer Systems 26.3 (2010): 318-323.

[2] Li, M., Fan, Y., Kim, M. K., Chen, H., and Lei, H.. *A scalable and elastic publish/subscribe service*. In IPDPS 2011, pp. 1254-1265.

[3] Gascon-Samson, J. et al. *Dynamoth: A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud*. In ICDCS 2015, pp. 486-496.

[4] Setty, V. et al. *Cost-effective resource allocation for deploying pub/sub on cloud*. In ICDCS 2014, pp. 555-566. IEEE, 2014.

[5] Fu, T. Z. J. et al. *DRS: Dynamic Resource Scheduling for Real-Time Analytics over Fast Streams*. In ICDCS 2015.

[6] Tran, N.-L., Skhiri, S., and Zimnyi, E. *Eqs: An elastic and scalable message queue for the cloud*. In CloudCom 2011, pp. 391-398.

[7] Fernandez, H., Pierre, G., Kielmann, T. *Autoscaling web applications in heterogeneous cloud infrastructures*. In IC2E 2014 (pp. 195-204).

[8] Nguyen, P., Nahrstedt K. *Resource Management For Elastic Publish Subscribe Systems: A Performance Modeling-Based Approach (Extended Version)*. http://hdl.handle.net/2142/89262. Online; accessed 6-May-2016.

[9] Urgaonkar, B. et al. *An analytical model for multi-tier internet services and its applications*. In SIGMETRICS 2005.

[10] Ali-Eldin, A., Tordsson, J., Elmroth, E. *An adaptive hybrid elasticity controller for cloud infrastructures*. In NOMS 2012 (pp. 204-212).

[11] Ali-Eldin, A. et al. *Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control*. In ScienceCloud 2012 Workshop (pp. 31-40).

[12] Jacobsen, H.A. et al. *The PADRES Publish/Subscribe System*. Principles and Applications of Distributed Event-Based Systems 2010, 164, p.205.

[13] Li, G., & Jacobsen, H. A. (2005). Composite subscriptions in content-based publish/subscribe systems. *In Middleware 2005* (pp. 249-269). Springer Berlin Heidelberg.

[14] Bennani, M. N., Menasce, D. A. Resource allocation for autonomic data centers using analytic performance models. In ICAC 2005 (pp. 229-240).

[15] Bitran, G. R., Morabito, R. *Open Queueing Networks: Optimization and Performance Evaluation Models for Discrete Manufacturing Systems*. Production and Operations Management 1996.

[16] Urgaonkar, B. et al. *Agile dynamic provisioning of multi-tier internet applications*. In ACM TAAS 2008.

[17] Boxma, O. J., Kan, A. R., & van Vliet, M. (1990). Machine allocation problems in manufacturing networks. *European Journal of Operational Research*, 45(1), 47-54.

[18] van Vliet, M., Kan, A. H. R. (1991). Machine allocation algorithms for job shop manufacturing. *Journal of Intelligent Manufacturing*, 2(2), 83-94.

[19] Dyer, M. E., Proll, L. G. On the validity of marginal analysis for allocating servers in M/M/c, 1019-1022.

[20] Kameda, H., Zhang, Y. (1995). Uniqueness of the solution for optimal static routing in open BCMP queueing networks. *Mathematical and Computer Modelling*, 22(10), 119-130.

[21] Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec, A.M., 2003. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2), pp.114-131.

[22] Rowstron, A. et al. *SCRIBE: The design of a large-scale event notification infrastructure*. In Networked group communication 2001, pp. 30-43, Springer Berlin Heidelberg.

[23] Rosenblum, D. S., and Wolf, A. L. (1997). A design framework for Internet-scale event observation and notification (Vol. 22, No. 6, pp. 344-360). ACM.

[24] Gupta, A., Sahin, O. D., Agrawal, D., and Abbadi, A. E. (2004). *Meghdoot: content-based publish/subscribe over P2P networks*. In Middleware 2004, pp. 254-273.

[25] I. Aekaterinidis and P. Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe dht network. *In ICDCS 2006*.

[26] Zaharia, M. et al. *Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling*. In EuroSys 2010, pp. 265-278.

[27] Melnik, S. et al. *Dremel: interactive analysis of web-scale datasets*. In VLDB Endowment 2010, 3(1-2), 330-339.

[28] Hindman, B. et al. *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*. In NSDI 2011, Vol. 11, pp. 22-22.

[29] Vavilapalli, V.K. et al. *Apache hadoop yarn: Yet another resource negotiator*. In SoCC 2013.

[30] Barker, A. and Van Hemert, J. *Scientific workflow: a survey and research directions*. In Parallel Processing and Applied Mathematics 2007, pp. 746-753, Springer Berlin Heidelberg.

[31] Tang, W. et al. *A scalable data analysis platform for metagenomics*. In BigData 2013, pp. 21-26.

[32] Dayal, U., Hsu, M. and Ladin, R., 2001, September. Business Process Coordination: State of the Art, Trends, and Open Issues. In VLDB 2001 (Vol. 1, pp. 3-13).

[33] Song, Z., Crdenas, A.A. and Masuoka, R., 2010, November. Semantic middleware for the Internet of Things. *In Internet of Things (IOT)*, 2010 (pp. 1-8). IEEE.

[34] Tijms, H.C., 1986. Stochastic modelling and analysis: a computational approach. *John Wiley & Sons, Inc.*.

[35] Whitt, W., 1993. Approximations for the GI/G/m queue. *Production and Operations Management*, 2(2), pp.114-161.