Computer Networks xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Scalable and elastic total order in content-based publish/subscribe systems

Xingkong Ma, Yijie Wang*, Xiaoqiang Pei, Fangliang Xu

Science and Technology on Parallel and Distributed Processing Laboratory, College of Computer, National University of Defense Technology, Changsha, Hunan 410073, PR China

A R T I C L E I N F O

Article history:
Received 5 June 2014
Received in revised form 16 March 2015
Accepted 6 April 2015
Available online xxxx

Keywords:
Total order
Publish/subscribe
Content-based
Data dissemination
Cloud computing

A B S T R A C T

Total order as a messaging guarantee scheme ensures that events sent to a set of subscribers are delivered by these subscribers in the same order. It has become increasingly important in content-based publish/subscribe (pub/sub) systems. Due to the large-scale live content and the churn workloads in the big data era, current emergency applications present a new challenge: how to provide a scalable and elastic total order service in content-based pub/sub systems. Most existing total order approaches cannot adapt to the churn workloads, and generate high delivery latency in the face of high arrival rate of live content. To this end, we propose a **s**calable and **e**lastic **t**otal **o**rder service, called SETO, for content-based pub/sub systems in the cloud computing environment. SETO uses a two-layer pub/sub framework to decouple the event matching service and the total order service. In this framework, events are forwarded to their interested subscribers by multiple parallel servers. Through a preceding graph building technique, non-conflicting events in the same server are allowed to be delivered simultaneously, which greatly reduces the delivery latency. The performance-aware provisioning technique in SETO elastically adjusts the scale of servers to adapt to the churn workloads. To evaluate the performance of SETO, tens of servers and thousands of subscribers are deployed in our CloudStack testbed. Extensive experiments confirm that SETO can linearly reduce the delivery latency with the growth of servers, adaptively adjust the scale of servers in less than 5 s, and significantly outperforms the state-of-the-art approaches under diverse parameter settings.

1. Introduction

Publish/subscribe (pub/sub) model has received increasing attention to build large-scale distributed systems through decoupling senders and receivers in space, time and synchronization [1]. In publish/subscribe systems, subscribers express their interests in the form of *subscriptions*, and publishers publish their live content in the form of *events*. Thus, the system matches events against subscriptions and forwards events to the interested subscribers.

Current content-based pub/sub systems do not natively offer strong guarantees about the total order of events. Informally, total order ensures that events sent to a set of subscribers are delivered by all these subscribers in an uniform order. There are many pub/sub applications that require total order to ensure unavoidable concurrency. For instance, in network-centric warfares, the early warning radar aircraft sends operational instructions to multiple combat units. It would lead to fatal damage for

* Corresponding author. Tel.: +86 13308491230.
E-mail addresses: maxingkong@nudt.edu.cn (X. Ma), wangyijie@nudt.edu.cn (Y. Wang), xiaoqiangpei@nudt.edu.cn (X. Pei), xuflnk@gmail.com (F. Xu).

http://dx.doi.org/10.1016/j.comnet.2015.04.001
1389-1286/© 2015 Elsevier B.V. All rights reserved.

these units to execute uncooperative operations due to differences in the order they receive instructions. In distributed coordination service [2], multiple update operations pointing to the same object may arrive at the servers simultaneously. It would bring inconsistent views among replicated servers if these servers execute update operations in different orders. Another set of examples that need total order are for fairness. For instance, players in distributed online games [3] receive publications from each other to update their states, such that they have the consistent game views. It would be unfair for players to make decisions if they receive publications in different orders. Similarly, investors in the stock market are notified the updates of trade volumes and prices [4]. It is necessary to receive the same order of updates for investors. Otherwise, different stock market views caused by out-of-order updates lead to unfair trades.

In the big data era, providing total order primitive in content-based pub/sub systems is becoming a challenging problem. Firstly, the high arrival rate of live content requires the total order algorithm to be extremely scalable. For instance, NWS [5] provides up-to-the-minute weather information to subscribers in any region in America. As the arrival rate of live content increases, an inefficient total order algorithm would lead to a high delivery latency to subscribers, which makes the publications out-of-date. Secondly, the churn workload requires the total order algorithm to provide elastic schemes to reach a high performance price ratio. *Churn workload* mean that the arrival rate of events may frequently change in many emergency applications. For instance, in a smart transportation system [6], millions of messages are generated during peak hours, while few number of messages are generated during off-peak hours. In this scenario, it would either lead to low delivery throughput during peak hours or waste computing resource during off-peak hours if a fixed number of servers are deployed.

Existing total order algorithms offer diverse ordering semantics. However, most of them are not applicable to our context. This mainly stems from the following two reasons. The first reason is that many total order algorithms are designed for a single destination group [7–13] or a limited number of explicit destination groups [14–17]. In contrast, given n subscriptions, there are theoretically 2^n groups in content-based pub/sub systems. It will lead to large traffic overhead and high delivery latency if we adopt these algorithms to manage the exponentially increasing groups. The second reason is that existing total order algorithms in content-based pub/sub systems [18–21] do not scale well with the increasing arrival rate of live content. Events in these algorithms need to traverse a specific network to be matched and synchronized before being delivered, which brings a high delivery latency as a large number of events arrive at the system. Besides, existing content-based pub/sub systems [18–21] have no financial incentive to provide elastic total order service, since the peers in these systems are commonly autonomous users and they decide their own behaviors (joining or leaving) freely.

Recently, cloud computing provides great opportunities for real-time and reliable communication, and complex computing. The cloud computing environment offers infrastructure, platform and applications as services to one or more tenant organizations, where the servers are equipped with powerful computing capabilities and organized into high speed local networks. Many pub/sub systems [22–26] attempt to provide scalable and elastic service in the cloud platform. However, these systems focus on high event matching throughput, rather than the total order among events.

Motivated by these factors, this paper presents a scalable and elastic total order service for content-based pub/sub systems in the cloud computing environment, called SETO. From the perspective of the system, the framework of SETO consists of two layers: the matching layer and the delivery layer. The matching layer is responsible for matching events against subscription and sending events with their matched subscribers to the delivery layer. At this layer, we employs SEMAS [23] to implement high matching throughput. In SEMAS, through a hierarchical multi-attribute space partition technique (called HPartition), the content space is divided into multiple hypercubes, each of which is managed by one server. Subscriptions and events falling into the same hypercube are matched with each other, such that the matching latency can be greatly reduced. Besides, a performance-aware detection technique (called PDetection) is proposed in SEMAS to adaptively adjust the scale of servers based on the churn workloads.

The delivery layer is responsible for total ordering events and delivering them to their interested subscribers. The main novelty of this layer lies in a preceding graph building technique (called PGBuilder) and a performance-aware provisioning technique (called PProvision). The first aims to reduce the total order latency in a scalable manner. In PGBuilder, subscribers are divided into multiple groups, each of which is managed by a single server. That is, all servers of PGBuilder are able to detect total order conflicts among events simultaneously, which greatly reduces the delivery latency. Each server of PGBuilder constructs preceding graphs among arrival events, which can quickly detect non-conflicting events and deliver them in a parallel manner. Besides, to ensure reliable delivery, PGBuilder provides a series of dynamics maintenance mechanisms.

The second aims to achieve elastic total order service. Specifically, PProvision adjusts the scale of servers based on the churn workloads. Through a linear extrapolation method, each server estimates its own performance of total ordering. When the performance of the worst server is out of expectation, a number of servers are added or removed from the system to reach a high performance price ratio. The main novelty of PProvision is that it adjusts the scale of servers according to the dynamic proportional relation between the scale of servers and the performance of the worst server, which can greatly reduce the reconfiguring latency. In contrast, PDetection of SEMAS adds or removes servers one by one, which may lead to cascade provisioning.

Moreover, we design and implement a prototype on our CloudStack testbed. Extensive experiments based on a CloudStack testbed verifies that SETO can linearly reduce