Universal Cloud Computing

Abstract—Integration of applications, data-centers, and programming abstractions in the cloud-of-clouds poses many challenges to system engineers. Different cloud providers offer different communication abstractions, and applications exhibit different communication patterns. By abstracting from hardware addresses and lower-level communication, the publish/subscribe paradigm seems like an adequate abstraction for supporting communication across clouds, as it supports many-to-many communication between publishers and subscribers, of which one-to-one or one-to-many can be viewed as special cases. In particular, content-based publish/subscribe (CPS) systems provide an expressive abstraction that matches well with the key-value pair model of many established cloud storage and computing systems, and decentralized overlay-based CPS implementations scale up well. However, CPS systems perform poorly at small scale, e.g., one-to-one or one-to-many communication. This holds especially for multi-send scenarios which we refer to as entourages that may range from a channel between a publisher and a single subscriber to a broadcast between a publisher and a handful of subscribers. These scenarios are common in cloud computing, where cheap hardware is exploited for parallelism (efficiency) and redundancy (fault-tolerance). With CPS, multi-send messages go over several hops before their destinations are even identified via predicate matching, resulting in increased latency, especially when destinations are located in different data-centers or zones. Topic-based publish/subscribe (TPS) systems support communication at small scale more efficiently, but still route messages over multiple hops and inversely lack the flexibility of CPS systems. In this paper, we propose CPS protocols for cloud-of-clouds communication that can dynamically identify entourages of publishers and corresponding subscribers. Our CPS

protocols dynamically connect the publishers with their entourages through uberlays€. These uberlays€ can transmit messages from a publisher to its corresponding subscribers with low latency. Our experiments show that our protocols make CPS abstraction viable and beneficial for many applications. We introduce a CPS system named Atmosphere that leverages out CPS protocols and illustrate how Atmosphere has allowed us to implement, with little effort, versions of the popular HDFS and ZooKeeper systems which operate efficiently across data-centers.

1 INTRODUCTION

The advent of cloud brokers for mediating between dif-ferent cloud providers, and the cloud-of-clouds para-digm denoting the integration of different clouds— including clouds offering different abstractions (e.g., infra-structure as a service versus platform as a service)—pose new challenges to software developers. Indeed, while sup-port for developing specific types of applications to run in different individual cloud infrastructures is slowly becom-ing established, there is little support for programming applications that run across several clouds or types of clouds.

1.1 Cross-Cloud Communication

One particular aspect of such integration, addressed herein, is communication. The cloud-of-cloud paradigm postulates integration of multiple data-centers, cloud abstractions, and applications. Can a single communication middleware ful-fill all different arising needs for communication? In partic-ular, a candidate middleware system must be able to

R1. support a variety of communication patterns e.g.,communicationrate,numberof interating entities)effectively.

Given the variety of target applications (e.g., social networking, web servers), the system must be able to cope with one-to-one communica-tion as well as different forms of multicast (one-to-many, many-to-many). In particular, the system must be able to scale up and down (“elasticity”) based on current needs [27] such as number of com-municating endpoints;

R2. run on standard “low-level” network layers and abstractions without relying on any specific pro-tocols such as IP Multicast [17] that may be deployed in certain clouds but not in others or across clouds [36];

R3. provide an interface which hides cloud-specific hardware addresses and integrates well with abstractions of widespread cloud storage and com-puting systems in order to support a wide variety of applications;

R4. operate efficiently despite varying network latencies within/across datacenters.

1.2 Publish/Subscribe for the Cloud

One candidate abstraction is publish/subscribe. Compo-nents act as publishers of messages, and dually as subscrib-ers by delineating messages of interest. Examples of publish/subscribe services designed for and/or deployed in the cloud include Amazon's simple notification service (SNS) , LinkedIn's Kafka, Hadoop Hedwig, or Blue Dove . Intuitively, publish/subscribe is an adequate abstraction for cross-cloud communication because it supports generic many-to-many interaction, shields applications from specific lower-level communica-tion—in particular hardware addresses—thus supporting application interoperability and portability. In particular, content-based publish/subscribe (CPS) promotes an addressing model based on message proper-ties and corresponding values (with subscribers delineat-ing values of interest for relevant properties) which matches well the key-value pair abstractions used by many cloud storageb and computing frameworks.

1.3 Limitations

However, existing publish/subscribe systems for the cloud are not designed to operate beyond single datacen-ters, and CPS systems focus on scaling up to large numbers of subscribers: to "mediate" between published messages and subscriptions, CPS systems typically employ an over-lay network of brokers, with filtering happening down-stream from publishers to subscribers based on upstream aggregation of subscriptions. When messages from a pub-lisher are only of interest to one or few subscribers, such overlay-based multi-hop routing (and filtering) will impose increased latency compared to a direct multi-send via UDP or TCP from the publisher to its subscribers. Yet such scenarios are particularly wide-spread in third-party computing models, where many cheap resources are exploited for parallelism (efficiency) or redundancy (fault-tolerance). A particular example are distributed file-systems, which store data in a redundant manner to deal with crash failures , thus leading to frequent communi-cation between an updating component and replicas. Another example for multi-sends are (group) chat sessions in social networks.

Existing approaches to adapting interaction and com-munication between participants based on actual commu-nication patterns are agnostic to deployment constraints such as network topology. Topic-based publish/subscribe (TPS) where messages are published to topics and delivered to consumers based on topics they subscribed to—is typically implemented by assigning topics to nodes. This limits communication hops in multi-send scenarios, but also the number of subscribers. In short, CPS appealing, generic, communication abstraction (R2, R3), but existing implementations are not efficient at small scale (R1), or, when adapting to application characteristics, do not consider deployment constraints in the network (R4); inversely, TPS is less expressive than CPS, and existing systems do not scale up as well.

1.4 Atmosphere

This paper describes Atmosphere, a middleware solution that aims at supporting the expressive CPS abstraction across datacenters and clouds in a way which is effective for a wide range of communication patterns. Specifically, our goal is to support the extreme cases of communication between individual publisher-subscriber pairs (unicast) and large scale CPS, and to elastically scale both up and
down between these cases, whilst providing performance which is comparable to more specialized solutions for individual communication patterns. This allows applica-tions to focus on the logical content of communication rather than on peer addresses even in the unicast case: application components need not contain hardcoded addresses or use corresponding deployment parameters, as the middleware automatically infers associations between publishers and subscribers from advertisements and subscriptions.

Our approach relies on a CPS-like peer-based overlay network which is used primarily for "membership" pur-poses, i.e., to keep participants in an application connected, and as a fallback for content-based message routing. The system dynamically identifies clusters of publishers and their corresponding subscribers, termed entourages while taking network topology into account. Members of such entourages are connected directly via individual overlay
networks termed uberlays€, so that they can communicate
with low latency. The uberlay€ may only involve publishers and subscribers or may involve one or many brokers depending on entourage characteristics and resource avail-abilities of involved publishers, subscribers, brokers, and network links. In any case, these direct connections which are gradually established based on resource availabilities, will effectively reduce the latency of message transfers from publishers to subscribers.

1.5 Contributions

Several CPS systems have been proposed in the literature, and Atmosphere adopts several concepts presented therein. Thus, in the present paper, we focus on the following novel contributions of Atmosphere:

1) a protocol to dynamically identify topic-like entour-ages of publishers in a CPS system. Our technique hinges on precise advertisements. To not hamper flexibility, advertisements can be updated at runtime;
2) a protocol to efficiently and dynamically construct uberlays€ interconnecting members of entourages with low latency based on resource availabilities;
3) the implementation of a scalable fault-tolerant CPS system for geo-distributed deployments named Atmosphere that utilizes our entourage identifica-tion and uberlay€ construction techniques;
4) an evaluation of Atmosphere using real-life applica-tions, including social networking, news feeds, the HDFS distributed file-system, and the ZooKeeper distributed lock service demonstrating the effi-ciency and versatility of Atmosphere through perfor-mance improvements over more straightforward approaches. In particular, we describe how Atmo-sphere was instrumental in building, without much effort, versions of HDFS and ZooKeeper operating efficiently across datacenters.

3 ENTOURAGE COMMUNICATION

In this section, we introduce our solution for efficient com-munication between publishers and "small" sets of sub-scribers on a two-level geo-distributed CPS network of brokers with hierarchical deployments within individual regions as outlined in Fig. 2 for two regions. This solution can be adapted to existing overlay-based CPS systems char-acterized in Section 2.2.

3.1 Broker Hierarchies

Consider the number of publishers in region r to be P_r and there to be m^p_r subscribers in region r interested in some mes-sages from a given publisher p. Additionally, assume p to be publishing messages to G at rate f_p and the average size of these messages to be s_p.

Also take W_p to be the maximum bandwidth that a pub-lisher p has at disposition when transferring messages to a remote region, and the percentage utilization of the proces-sor (CPU) of p due to message transmission to be U_p. It is to be noted that while inter-region links usually have ample amounts of bandwidth, most of the time the user will be set-ting the value of W_p based on the economical factors (e.g., for the inter-region data transfer costs) instead of actual bandwidth limitations. Table 1 lists definitions of some of the notations used in this paper for easy reference.

3.2 Entourages

The range of messages published by a publisher p are iden-tified by advertisements t_p, which, as is customary in CPS, include the keys and the value ranges for each key. Analo-gously, the interest range of each subscriber or broker n is denoted by t_n. $t_p \setminus t_n$ denotes the common interest between a publisher p and a subscriber or broker n.

2 BACKGROUND

This section presents background information and the model of system considered.

We define the interest match between a publisher p and a subscriber/broker n as a numerical value that represents the fraction of the publisher's messages that the subscriber/ broker is interested in assuming the publisher to have an equal probability of publishing a message with any given value within its range.

3.3 Solution Overview

Next we describe our solution to efficient cross-cloud com-munication in entourages. The solution consists of three main parts which we describe in turn.

1) A decentralized protocol that can be used to identify entourages in a CPS system.
2) A mechanism to determine the maximum number K_p of direct connections a given publisher p can maintain without adversely affecting message transmission.
3) A protocol to efficiently establish auxiliary networks termed uberlay€s between publishers and their respec-tive subscribers.and the rest of the network could also become a bottleneck if messages are significantly large, or transmitted at a signifi-cantly high rate. This is particularly valid in a multi-tenant cloud environment. The transport protocols used by the publisher and latencies to the receivers could limit the rate at which the messages are transmitted.

3.4 Factors and Challenges

Capabilities of any node connected to a broker network are limited by a number of factors. A node obviously has to spend processor and memory resources to process and pub-lish a stream of messages. The bandwidth between the node and the rest of the network could also become a bottleneck if messages are significantly large, or transmitted at a signifi-cantly high rate. This is particularly valid in a multi-tenant cloud environment. The transport protocols used by the publisher and latencies to the receivers could limit the rate at which the messages are transmitted.

4 ATMOSPHERE

In this section, we describe Atmosphere, our CPS frame-work for multi-region deployments which employs the DCI protocol and overlay establishment introduced previously. The core implementation of Atmosphere in Java has approximately 3; 200 lines of code.

4.1 Overlay Structure

Atmosphere uses a two-level overlay structure based on broker nodes. Every application node that wishes to com-municate with other nodes has to initially connect to one of the brokers which will be identified as the node's par-ent. A set of peer brokers form a broker group. Each bro-ker in a group is aware of other brokers in that group. Broker-groups are arranged to form broker-hierarchies. Broker-hierarchies are illustrated. As the figure depicts, a broker-hierarchy is established in each consid-ered region. A region can typically represent a LAN, a datacenter, or a zone within a datacenter. At the top (root) level broker-groups of hierarchies are connected to each other. The administrator has to decide on the num-ber of broker-groups to be formed in each region and the placement of broker-groups.

Atmosphere employs subscription summarization to route messages. Each broker summarizes the interests of its subordinates and sends the summaries to its parent broker. Root-level brokers of a broker-hierarchy share their sub-scription summaries with each other. At initiation, the administrator has to provide each root-level group the iden-tifier of at least one root-level broker from each of the remote regions.

4.2 Fault Tolerance and Scalability

Atmosphere employs common mechanisms for fault toler-ance and scalability. Each broker group maintains a strongly consistent membership, so that each broker is aware of the live brokers within its group. A node that needs to connect to a broker-group has to be initially aware of at least one live broker (which will become the node's parent). Once connected, the parent broker pro-vides the node with a list of live brokers within its broker-group and keeps the node updated about membership changes. Each broker, from time to time, sends heartbeat messages to its children.

If a node does not receive a heartbeat from its parent for a predefined amount of time, the parent is presumed to have failed, and the node connects to a different broker of the same group according to the last membership update from the failed parent. A node that wishes to leave, sends an unsubscription message to its parent broker. The parent removes the node from its records and updates the peer brokers as necessary.

he number of levels of the broker-hierarchy. Nodes may subscribe to a broker at any level.

Atmosphere can be scaled both horizontally and verti-cally. Horizontal scaling is achieved by adding more brokers to groups. Additionally, Atmosphere can be vertically scaled by increasing

4.3 Flexible Communication

Atmosphere implements the DCI protocol of Section 3. To this end, each publisher sends COUNT messages to its broker. These messages are propagated up the hierarchy and once the robrokers are reached, are distributed to the remote regions to identify entourages. Once suitable

entourages are identified, uberlays€ are established which are used to disseminate messages to interested subscrib-ers with low latency.

When changes in subscriptions (e.g., joining/leaving of subscribers) arrive at brokers these may propagate corre-sponding notifications upstream even if subscriptions are covered by existing summaries; when arriving at brokers involved in direct connections these can notify publishers directly of changes, prompting them to re-trigger counts.

Fig. 5 shows the major entities of a Atmosphere deploy-ment and corresponding interactions that result in messages being exchanged through a direct connection.

4.4 Advertisements

By wrapping it with the client library of Atmosphere the DCI protocol for publishers/subscribers is transparent to application components, at the exception of advertisements which publishers can optionally issue to make effective use of direct connections.

Advertisements are supported in many overlay-based CPS systems, albeit not strictly required. Similarly, publish-ers in Atmosphere are not forced to issue such advertise-ments in Atmosphere, although effective direct connection establishment hinges on accurate knowledge of publication ranges. Atmosphere can employ runtime monitoring of published messages if necessary. For such inference, the cli-ent library of Atmosphere compares messages published by a given publisher against the currently stored advertisement
and adapts the advertisement if required. When witnessing significant changes, the new advertisement is stored and the DCI protocol is re-triggered.

Note that messages beyond the scope of a current adver-tisement are nonetheless propagated over the direct connec-tions in addition to the overlay. The latter is necessary to deal with joining subscribers in general as mentioned, while the former is done for performance reasons—the directly connected nodes might be interested in the message since the publisher's range of publications announced earlier can be a subset of the ranges covered by any subscriptions.

The obvious downside of obtaining advertisements only

by inference is that uberlay€ creation is delayed and thus latency is increased until the ideal connections are estab-lished. To avoid constraining publishers indefinitely to previously issued advertisements, the Atmosphere client library offers API calls to issue explicit advertisement updates. Such updates can be viewed as the publisher-side counterpart of parametric subscriptions whose native support in a CPS overlay network have been shown to not only have benefits in the presence of changing subscrip-tions, but also to improve upstream propagation of changes in subscription summaries engendered via unsubscriptions and new subscriptions.

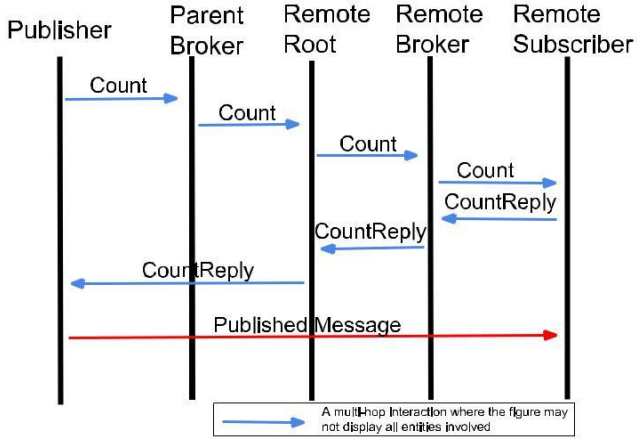
5 .RELATED WORK

This section elaborates on further related work and differen-ces to Atmosphere.

Chand and Felber introduce another CPS architecture for distributed systems. The approach is similar to Siena but has support for dynamic subscription and cancellation of interests. This is done by keeping record of the sub-interface through which a filter was received and not just the filter itself. The employed protocol ensures perfect routing which means that a message will be received by all the interested parties and no others. HyperCBR introduces a partition-based approach for content-based routing. Sub-scriptions and events are propagated in a multidimensional space. The system is partitioned such that every subscrip-tion partition interacts with at least one node of every mes-sage partition and vice versa. This model can be extended by adding more subscription dimensions. The authors use simulations and analytical studies to show that the model can scale up to a large number of nodes.

Many adaptive communication systems have been proposed in the literature. Yoneki and Bacon propose an adaptive publish/subscribe system suitable for mobile ad-hoc networks that represents summarized subscriptions in bloom filters. Rodrigues etal.introduce a mechanism to adapt gossip-based broadcast algorithms according to the amount of resources available to nodes and the global level of congestion. These adaptions focus on scaling up.

Among the first efforts to offer content-based address-ing as core network service is data-oriented network archi-tecture (DONA). The proposal focuses on encoding of content and content-addressing into low-level network packets. DONA has a broader scope than the present work, but does not address scaling up and down. Content-centric networking (CCN) is a related thrust. Its scope is yet more broadly defined.



5.1 Other Solutions for Cloud Communication

Cloud service providers such as Microsoft and Amazon have introduced content delivery networks (CDNs) for com-munication between their datacenters. Microsoft Azure CDN caches Azure blob content at strategic locations to make them available around the globe. A special variable, TTL, decides the time after which the cached blob content should expire. Therefore the content requests that come only once during a TTL amount of time will not have any performance and cost advantages over a request that is always retrieved remotely. Amazon's CloudFront is a CDN service that can be used to transfer data across Amazon's datacenters. CloudFront can be used to transfer both static and streamed content using a global network of edge loca-tions. Objects are organized according to unique domain names and when an end-user requests an object, it is auto-matically transferred to the nearest edge location. CDNs focus on stored large multimedia data rather than on live communication. In addition, both above-mentioned CDN networks can be used only within their respective service provider boundaries and individual regions.

Volley strategically partitions geo-distributed stored data so that the individual data items are placed close to the global "centroid" of the past accesses. For this Volley extracts information about users and data item accesses from log files of the application and needs to be configured with other information such as the number of datacenters, their geographic location and a policy to determine cost of communication between two given datacenters. Volley peri-odically runs an iterative algorithm to determine the opti-mum location for data items and places them in a datacenter close to this location depending on space avail-abilities. The global resource allocation problem for geo-dis-tributed cloud-based applications addressed by solutions such as Volley is orthogonal to the problem of efficient com-munication within geo-distributed cloud applications which we focus on. The solutions are complementary.

Use of IP Multicast has been restricted in some regions and across the Internet due to difficulties arising with multi-cast storms or multicast DOS attacks. Dr. Multicast [36] is a protocol that can be used to mitigate these issues. The idea is to introduce a new logical group addressing layer on top of IP Multicast so that access to physical multicast groups and data rates can be controlled with an acceptable user pol-icy. This way system administrators can place caps on the amount of data exchanged in groups and the members that can participate on a group. Dr. Multicast specializes on intra-datacenter communication and does not consider inter-datacenter communication.

6 CONCLUSIONS

We are currently working on complementary techniques that will further broaden the range of efficiently supported communication patterns, for example the migration of sub-scribers between brokers guided by resource usage on these brokers. Additionally we are exploring the use of Atmo-sphere as the communication backbone for other systems including our framework for efficiently executing Pig/ PigLatin workflows in geo-distributed cloud setups and our G-MR system for efficiently executing sequences of MapReduce jobs on geo-distributed data sets.

ACKNOWLEDGMENTS

The authors are very grateful to Amazon and to Larry Peter-son for making it possible to evaluate our research in EC2 and VICCI respectively. This work is financially supported by DARPA Grant N11AP20014 “Large-Scale Cloud-based Data Analysis”, Purdue Research Foundation Grant 204533 “Seamless Cloud Computing”, Google Research award “Geo-Distributed Big Data Processing”, and Cisco Research award “A Fog Architecture”. This paper extends our prior work published at Middleware 2013 .

REFERENCES

[1] Active MQ, http://activemq.apache.org/
[2] Amazon Simple Notification Service, http://aws.amazon.com/ sns/
[3] Apache HDFS, http://hadoop.apache.org
[4] Apache ZooKeeper, http://hadoop.apache.org/zookeeper
[5] Cleverbot, http://zookeeper.apache.org
[6] Websphere MQ, http://www-01.ibm.com/software/integration/ wmq/