IEEE CLOUD COMPUTING

VOLUME 5, NUMBER 2 MARCH/APRIL 2018

Navigating the Cloud

IEEE IEEE computer society IEEE ComSoc IEEE Communications Society

www.computer.org/cloud

IEEE CLOUD COMPUTING

www.computer.org/cloudcomputing

EDITORIAL OFFICE
Publications Coordinator:
cloudreview@allenpress.com
Authors: www.computer.org/web/peer-review/magazines
Letters to the Editors: bbrannon@computer.org
Subscribe: www.computer.org/subscribe
Subscription change of address:
address.change@ieee.org
Missing or damaged copies:
help@computer.org
Reprints of articles: cloud@computer.org
IEEE Cloud Computing
c/o IEEE Computer Society
10662 Los Vaqueros Circle,
Los Alamitos, CA 90720 USA
Phone +1 714 821 8380; Fax +1 714 821 4010
www.computer.org/cloud-computing

IEEE ComSoc IEEE Communications Society

SUSTAINABLE FORESTRY INITIATIVE Certified Sourcing www.sfiprogram.org SFI-01681

IEEE Cloud Computing: (ISSN 2325-6095) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe: Go to www.computer.org/subscribe for more information on subscribing.

IEEE CLOUD COMPUTING

March/April 2018
Vol. 5, No. 2

www.computer.org/cloud

TABLE OF CONTENTS

Feature Articles

Columns and Departments

Also in This Issue

ISSN: 2325-6095 Copublished by the IEEE CS and IEEE ComSoc

Clouds for Science Tackle Challenges Facing Industry and Society

Keith Jeffery
Keith G. Jeffery Consultants

Editor:
Mazin Yousif,
mazin@computer.org

Since around 1950, scientific challenges have been an important application domain for computing and thus a significant factor driving developments in hardware and software. Almost 20 years ago, this led to the idea of grid computing, a key progenitor of today's cloud computing. Grid initiatives primarily focused on linking up supercomputing resources in North America and the Far East; in Europe, the focus was more on sharing distributed computing and information resources.

Scientific challenges driving developments in computing range from the physical to the social sciences and include understanding the fundamental properties of matter; understanding our place in the universe and how we got here through studies ranging from molecular evolution to astronomy; finding mechanisms to improve human longevity and lifestyle ranging from DNA sequencing to epidemiology; improving weather forecasting for both agriculture and aviation; understanding climate change and its effects physically, economically, and socially; prediction and management of economic trends in financial markets; and more. While some challenges are "blue sky" and curiosity-led, most are significant for commerce and industry and hence wealth creation, while others are significant for government policies and hence improvement in the quality of life. It is for these reasons that the spheres of science, computing, industry and government are coalescing. A variety of initiatives are underway globally, but the EC (European Commission) H2020 (Horizon 2020) program supporting research and innovation (https://ec.europa.eu/programmes/horizon2020/en/news/horizon-2020-work-programme-2018-2020) is of particular interest.

Europe has a collection of RIs (research infrastructures) that support international communities in particular domains such as particle physics (the well-known CERN laboratory), space science, materials science, biomedical science, environmental science, social science, arts and humanities, and so on. These are cataloged in the European Strategy Forum on Research Infrastructures roadmap (www.esfri.eu/roadmap-archive), which advises the EC on funding priorities. These RIs support hundreds of thousands of researchers worldwide. A large part of their support is

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

through ICT (Information and Communication Technology), allowing users to identify and access assets (such as datasets, software services, workflows, sensor networks, computing resources, expert advice, laboratory equipment, and so on), construct workflows, and execute them to obtain research results. The RIs provide a focus for each domain community. This is very much the environment of large numbers: big data, high throughput (as well as high performance) computing on many execution platforms, and very large numbers of sensors within and outside of laboratories.

In parallel to these RIs there is an ICT research program (https://ec.europa.eu/programmes/horizon2020/en/area/ict-research-innovation) focused on societal challenges that spans HPC (high performance computing), the IoT (Internet of Things) and cloud computing. In particular, it addresses improved software development methods together with software development platforms and tools to generate applications that meet user requirements, are effective and efficient, and can be deployed flexibly—even during execution—across multiple heterogeneous execution platforms.

The environment of domain-specific research and computing science research is a fertile ground for novel ideas and techniques. Perhaps most famously, the Worldwide Web came from CERN. Domain research requirements pose challenges that can be solved by advanced ICT and thus, joint teams across the two areas of research are increasingly cooperating to meet those challenges. The aim is for better and more advanced domain research and in particular cross-domain research addressing societal challenges such as the 19 UN world issues (www.un.org/en/sections/issues-depth/global-issues-overview) or the seven societal challenges of the EU (European Union; https://ec.europa.eu/programmes/horizon2020/en/h2020-section/societal-challenges). It is also expected—and indeed intended—that innovation in this field will translate to commercialization and thus wealth creation as well as evidence-led government policy-making.

There is a large market for advanced computing, including cloud computing, to support research computing in pharmaceuticals, aerospace, oil exploration and other industries. Increasingly, research teams include both academic and commercial partners, working together to ensure the results are applicable for—and translate directly to—wealth creation and/or quality of life improvements.

The major ICT/computer science challenges include:

1. **Heterogeneity**: Integrating the metadata description of heterogeneous assets (including multilingualism) to form a common catalog for asset access.
2. **Availability**: Ensuring availability of the assets, i.e., curation, and involving media migration and appropriate licensing, typically open source, with appropriate metadata for discovery.
3. **Contextualization**: Ensuring assets are of sufficient relevance and quality for the purpose, which in turn requires rich metadata.
4. **Locality**: Managing data locality. For example, should a portion or entirety of a dataset be moved to another host or replicated at another host to optimize execution, considering processing time and data transport costs, including latency? Should we move the code to the data or the data to the code – especially if the data is heterogeneous and distributed?
5. **Trust, privacy and security**: Dealing with privacy and security issues including AAAI (Authentication, Authorization, Accounting Infrastructure) and the use of data locality and partitioning (as well as/instead of encryption) for security and privacy.
6. **Provenance**: Managing versioning, currency of replicas, and tracking the evolution of assets.
7. **Resources**: Knowing the availability and status of execution platforms described by metadata.
8. **Workflow enablement**: Optimizing deployment of the application workflows across heterogeneous execution platforms including HPC, IoT, and clouds.

This eighth challenge involves assessment of parameters for a particular application execution covering all the previous challenges and mapping this "demand" to the "supply" of various combinations of execution platforms, which may include Fog/Edge computing and IoT in the sensor

networks or laboratory equipment, HPC for large-scale analytics or simulation, and public, hybrid, or multi-cloud computing to provide computing resources required in excess of that available within the RIs.

Each of these challenges is difficult and there has been extensive research on all of them over many years. It has become clear that unless these challenges are resolved, progress will stall not only in open science but also in its related economic and societal domains. However, there is progress. Federated cloud platforms are emerging to provide homogeneity of computing resource provisioning over heterogeneous platforms. In Europe, Helix-Nebula (www.hnscicloud.eu), initiated by CERN in 2011, is a public-private partnership providing federated cloud services for science. IEEE began the P2302 initiative (http://sites.ieee.org/sagroups-2302) in 2011 to provide Intercloud, while NIST (www.nist.gov) has a Federated Cloud Public Working Group (www.federalregister.gov/documents/2017/08/30/2017-18354/federated-cloud-public-working-group). The IEEE Cloud Computing Standards Committee (www.computer.org/web/standards/cloud) also joined these initiatives in August 2017.

In addition, EC is funding a project named EOSC: European Open Science Cloud (https://ec.europa.eu/research/openscience/index.cfm?pg=open-science-cloud). The idea is to apply the cloud concept to encourage easier utilization (including reuse) of ICT to support research activity by sharing of assets: put simply "put everything into the EOSC pot and stir vigorously to produce new science." Furthermore, the plan is for commercial organizations to avail themselves of this resource and to build new and exciting commercial services upon assets, especially datasets and software services, most of which are outputs from publicly-funded research. While these high-level aims are admirable, they imply a lot of detailed ICT engineering to make them a reality.

Several specific initiatives are of particular value. The PaaSage project (https://paasage.ercim.eu) demonstrated feasibility for application workflows that were characterized by the CAMEL (http://camel-dsl.org) language (now being integrated with TOSCA; http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html) and not to be confused with the Apache Camel language) deployed flexibly with execution monitoring results stored as metadata. MELODIC (www.melodic.cloud) builds upon PaaSage and takes into account several characteristics of the data, particularly data locality, which typically is the most important factor in optimizing deployment of an application workflow across multiple heterogeneous platforms.

So, the key aspect of this joint research is taking an end-user request, understanding it (including user context), utilizing the catalog(s) to construct a workflow, surveying availability and state of execution platforms, deploying the application, monitoring and adjusting (adaptation or re-deployment) to optimize execution, all while taking into account the challenges listed above. Cloud computing—in the widest sense—is a key component of this activity and the rapid advances in cloud computing are starting to make this dream realizable.

ABOUT THE AUTHOR

Keith Jeffery is a consultant specializing in technical coordination of ICST projects, knowledge and technology transfer, and research information systems. He received a PhD from the University of Exeter in Geology and Earth Science. Contact him at keith.jeffery@keithgjefferyconsultants.co.uk.

Mazin Yousif is the Editor in Chief of *IEEE Cloud Computing* magazine. Contact him at mazin@computer.org.

Who has your Cloud-Back?

Christine Miyachi
Xerox Corporation

Cloud computing security and legal issues have been in the news on a daily basis. How are industry, government, and individuals handling the risks?

For you late filers, the IRS (Internal Revenue Service—U.S. tax revenue collection agency) online system for filing taxes is broken. '"On my way over here this morning, I was told a number of systems are down at the moment,' IRS Acting Commissioner David Kautter told lawmakers at an IRS oversight hearing Tuesday [17 April 2018]. 'We are working to resolve the issue and taxpayers should continue to file as they normally would.'"[1] These systems are supported by cloud servers although the IRS appears need to update its technology.[2,3]

I normally file my taxes online. So how would I "normally" file now? Will the IRS give me an extension? Will I have to go back to paper but miss the deadline? Do I pay the penalty? Are there any protections for me?

Quick: Answer the above questions and don't take longer than a few seconds to do so. If your data backup system, or worse yet, your primary application running in the cloud stopped working tomorrow, what would happen?

What if your social media profile or cloud-based contacts disappeared? Could you get a job? Contact people?

What if there was a search warrant out for your data in the cloud? Could legal authorities get that data?

This column will explore the recent news about the data we store in the cloud knowingly and unknowingly, and whether or not we own that data. Although regulations vary across jurisdictions and continue to evolve, it is universally believed that an individual's data is a digital gold mine. How do we, as individuals, control our own individual pieces of data gold?

MOBILE APPLICATION DATA MINING

In the late nineties, I was taking a graduate class on internet programming using Perl and CGI. The professor asked us, "What if your spending patterns on the Web were used by a corporation to sell you things? Or by the government? Or by law enforcement? I personally didn't care. I had nothing to hide. He warned me saying that may not always be the case. One of the students wrote on a chat board: "I guess that I must be a very dull person but I do not care who knows where I

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

am and what I am buying. What are these other people doing that is so nefarious? I've looked at some bikini clad women on the Internet now and then—so what. Get a life people!"

It turns out that way more data about way more things is being collected about us on our phones than we could have ever imagined. AppCensus is a website that provides privacy ratings for over 80,000 Android apps.[2] Applications running on an Android device aren't supposed to obtain the device's unique code (referred to as an Android ID which identifies the phone's hardware[4]), because that could be used to track the phone's entire workings for the life of the phone—and therefore the duration of your use of that phone. All your phone's data, including your identity data, every time you did something on your phone when you were supposed to be doing something else, and much more, is being stored.

And it gets more disturbing. Serge Eglemen, the creator of AppCensus,[5] says that 2,500 apps (half of what he tested) have sensitive data that could help to identify the users without permission, which violates the Children's Online Privacy Protection Rule.[6]

So what to do? You can start by studying your privacy settings on your Google Dashboard[7] and be overwhelmed by all the data collected about you. Your trips, orders, searches, and photos you have taken. You can be befuddled by going into the Facebook settings for privacy.[8] In addition, you can read in full the privacy statements of every application you install on your phone. And even then, because applications sometimes don't follow their own policies, you still don't truly know what will happen to your data.

With the Facebook fiasco of 87 million users data being leaked to Cambridge Analytica without permission, Mark Zuckerberg, the Facebook CEO,[9] told the Senate that the company is going through "a broader philosophical shift in how we approach our responsibility as a company"[10] But several senators pointed out that Facebook is a monopoly and the alternatives are slim. A young adult I know deleted his Facebook account in high school only to realize in college that he couldn't live without it. Study groups, sports, and social gatherings were all done through the site. And he went through a pariah stage when he started back up because he had no connections.

This digital gold mine is your life. These applications are mining everything about you and the only way to prevent it is to go back to using a landline phone and stop using the Internet. But times are changing.

CLOUD STORAGE LEGAL PROTECTIONS

In Zuckerberg's remarks to the US Senate, he said that Facebook must give the user simple language regarding what his company does with their data and a way to permanently and quickly delete their personal data. In California, the state government is looking at creating an oversight organization that would require exactly this. Europe has already enacted this and I have noticed recently that groups I work with are scrambling to comply with the General Data Protection Regulation, which will go into effect in May.[5] The law says:

- Users must give explicit consent for use of their personal data, and can withdraw that consent at any time.
- Companies must provide copies of data on demand.
- If that data is stolen, companies must notify victims within 72 hours.
- Violations can result in fines of as much as 4 percent of a company's annual revenues, or 20 million euros — about $25 million USD.[11]

American companies might follow the European standard because companies find it easier to maintain a single process for ensuring protections, rather than having various protections based on regions.

"If we're going to see a move to do this kind of thing," said Danny O'Brien, a privacy activist at the Electronic Frontier Foundation, "right now is that moment."[12]

The Government Wants Your Data

A case that has now reached the Supreme Court is about Microsoft Corporation and its refusal to comply with a warrant seeking data. The data happens to be stored in Ireland and Microsoft says that the data is under Irish law and not US law. The Clarifying Overseas Use of Data (CLOUD) Act[13] will establish that US warrants cover data stored by US companies regardless of where the servers physically exist. At first, I thought that I didn't really want the government to get at my data. But consider the problem of data localization.

Some countries get around having a law like the CLOUD Act by requiring that data about their citizens be stored only in their country. China is pushing for data to be stored at their government-run data centers so they get unhindered access to data. Information uploaded by Chinese iCloud customers is stored at a Chinese government-run data center located in China.[13]

Your Data is Safe Up Here

If your storage on a cloud server became unavailable, do you have any rights or even any way to get it back? For now I'm keeping a local copy on a hard drive and a cloud-based copy of all of my important data. And I'm using multiple clouds. But even more reliable solutions may be on the way. One company is proposing what they say is a failsafe way to store data. Wasabi Technologies Inc. is working on data centers orbiting the earth called "Space Belt."[14] "It's super, super secure," Wasabi cofounder David Friend said. "To knock this thing out, you'd have to launch a bunch of satellite-killers."[15] The technology looks to be affordable and offers an alternative to traditional storage, although presumably, data access times might not be on par with terrestrial solutions

CONCLUSION

The safety of our personal information and all that we do in the cloud is up for grabs and we need to take control. Personal persistence[16] and better technology may be the answer. *IEEE Cloud Computing* magazine has many articles on security related issues[17] including a recent article that examine breakthroughs in blockchain to make user data more secure.[18] But government controls will also be necessary to make sure we are protected and elected officials are learning about the issue to make better laws. Our digital data is our essence, worth more than gold, and we need to protect ourselves.

REFERENCES

1. S. Walsh, "IRS Gives Taxpayers an Extra Day to File after Computer Crash," *The Boston Globe*, 2018; www.bostonglobe.com/news/politics/2018/04/17/irs-electronic-filing-system-breaks-down-hours-before-tax-deadline/nqhlDvuWSqNdGtTrQ5jpSL/story.html.
2. "Hey IRS, Get off My Cloud," *CPA Practice Advisor*, blog; www.cpapracticeadvisor.com/blog/10951491/hey-irs-get-off-my-cloud.
3. *U.S. Treasury Inspector General for Tax Administration (TIGTA)*, U.S. Treasury; www.treasury.gov/tigta.
4. *Best Practices for Unique Identifiers*, Android Developers, 2018; https://developer.android.com/training/articles/user-data-ids.html.
5. *AppCensus*; https://www.appcensus.mobi/.
6. "Children's Online Privacy Protection Rule," *ECFR - Code of Federal Regulations*, Federal Trade Commission; www.ecfr.gov/cgi-bin/text-idx?SID=4939e77c77a1a1a08c1cbf905fc4b409&node=16%3A1.0.1.3.36&rgn=div5.
7. "Sign In," *Google Accounts*, Google; https://myaccount.google.com/dashboard.
8. "Privacy Settings and Tools," Facebook; www.facebook.com/settings?tab=privacy.
9. N. Berger, "Facebook to Send Cambridge Analytica Data-Use Notices Monday," *The Boston Globe*, 2018; www.bostonglobe.com/business/2018/04/08/facebook-send-cambridge-analytica-data-use-notices-monday/Yv0IgzVaPosgJrG1sloHgK/story.html.

10. "Zuckerberg: We're Going through a Broader Philosophical Shift as a Company," *CNBC*, CNBC, 2018; www.cnbc.com/video/2018/04/10/zuckerberg-were-going-through-a-broader-philosophical-shift-as-a-company.html.
11. *EU GDPR Portal*; www.eugdpr.org.
12. N. Duarte, "Is It Time to Lay down the Law on the Internet?," *The Boston Globe*, 2018; www.bostonglobe.com/business/2018/03/26/time-lay-down-law-internet/dRaKgfT0S5h2UuLj64m8MO/story.html.
13. J. Botsford, "Should Foreign Police Be Able to Search for Evidence in US Cloud Data?," *The Boston Globe*, 2018; www.bostonglobe.com/business/2018/03/25/should-foreign-police-able-search-for-evidence-cloud-data/ruxEeff9hvL3nYXWZwTDXK/story.html.
14. *Wasabi*; www.wasabi.com.
15. H. Bray, "Data Storage beyond the Clouds: Wasabi Promises a Super-Secure System in Space," *The Boston Globe*, 2018; www.bostonglobe.com/business/2018/03/19/data-storage-beyond-clouds-wasabi-promises-super-secure-system-space/6MIGtLg6PRjcuaESIEWhxL/story.html.
16. C. Khanna, "Deleting Facebook Is Just a Start. Thousands of Apps Can Take Your Data," *The Boston Globe*, 2018; www.bostonglobe.com/business/2018/03/30/deleting-facebook-just-start-thousands-apps-can-take-your-data/Vtw1Cmjel7T6B5g2txGQ6H/story.html.
17. C. Davis, "Realizing Software Reliability in the Face of Infrastructure Instability," *IEEE Cloud Computing*, vol. 4, no. 5, 2017; doi.org/http://doi.ieeecomputersociety.org/10.1109/MCC.2017.4250927.
18. C. Esposito et al., "Blockchain: A Panacea for Healthcare Cloud-Based Data Security and Privacy?," *IEEE Cloud Computing*, vol. 5, no. 1, 2018; doi.org/http://doi.ieeecomputersociety.org/10.1109/MCC.2018.011791712.

ABOUT THE AUTHOR

Christine Miyachi is a systems engineer at Xerox Corporation and holds several patents. She works on Xerox's Extensible Interface Platform, which enables developers to create applications that work with Xerox devices by using standard web-based tools. Miyachi graduated from the University of Rochester with a BS in electrical engineering. She holds two MIT degrees: an MS in technology and policy/electrical engineering and computer science and an MS in System Design and Management. Contact her cmiyachi@alum.mit.edu.

Cloud Message Queueing and Notification: Challenges and (Blockchain) Opportunities

Christian Esposito
University of Napoli "Federico II"

Francesco Palmieri
University of Salerno

Kim-Kwang Raymond Choo
University of Texas at San Antonio

Editor:
Kim-Kwang Raymond Choo
raymond.choo@fulbrightmail.org

The current mobile app development practice, like other large-scale network-centric software projects, is characterized by the use of event notification facilities supporting the exchange of effective and efficient data flows between the application's front-end, usually located on customer's terminal equipment, and the back-end services available within the cloud. In order to avoid the need of setting up the notification infrastructure from scratch for any new application, many cloud service providers and mobile system manufacturing companies provide cloud-based messaging solutions. Such solutions, however, are often characterized by vulnerabilities that can be exploited to compromise the mobile applications' security by violating the privacy and integrity in their operations. In this work, we analyze these issues and posit the potential for the blockchain technology to mitigate such threats.

As noted in the first issue of this column,[1] cloud security and privacy are topics that span beyond any single domain / discipline. Therefore, to better reflect the broader, multidisciplinary nature of the security and privacy challenges in a cloud and related environments (for example, fog computing and edge computing), this column will be renamed 'Cloud Security and Privacy.'
–Kim-Kwang Raymond Choo

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

Event-driven communications are prevalent in network-centric applications where a large number of data sources and destinations need to exchange messages in a flexible and scalable manner.[2] Such a communication pattern is typically supported by publish/subscribe services, implemented as a federation of brokers queueing the incoming messages and routing them towards the interested destinations. Therefore, apart from the effort of implementing the communication logic within their applications, developers must design and deploy the brokers' federation by using their computing commodities. Such a task may be seen as a nuisance, as well as an unnecessary cost, particularly for small-scale software projects with a limited budget. However, it requires the fast prototyping/implementation of an idea to be concretely presented to potential investors. This may also result in an unnecessary effort duplication in those projects that have most of their backend components located within the cloud. Indeed, cloud computing can be effectively used in dealing with such an issue due to its elastic and pay-per-use resource provisioning model. A cloud infrastructure can host the brokers and make them available according to the Software as a Service (SaaS) paradigm to developers without worrying about their configuration, deployment and management, together with the relative operational efforts and costs. Such solutions are referred to as Cloud Messaging or Cloud-based Push Notification facilities,[3] and major cloud service providers have already put in place their own solutions that users can freely use, such as Firebase Cloud Messaging (FCM) by Google, Apple Push Notification Service, or Windows Push Notification. Such solutions have been thought within the context of mobile application (app) development[4] to provide flexible notification services between the apps and central servers, usually also located within the cloud, so as to cope with the high degrees of app engagement and user retention, by offering an always available and extremely flexible/scalable communication architecture as well as an improved usage experience to developers. In addition, the current research trends are leveraging these solutions to facilitate the design of Internet of Things (IoT) products, and to support the interaction of IoT nodes with the cloud or smartphones.[4]

CLOUD MESSAGING TO INTEGRATE SERVERS WITH SMARTPHONES AND IOT DEVICES

FCM is a representative example of a Cloud Messaging solution due to its increasing popularity for mobile app development within the Android platform.[6] The other solutions differ only for the specific communication protocol, message format or messaging product being adopted, but they share the same key architecture and programming principles.

A mobile app consists of a central server that collects and processes data coming from multiple instances of client applications running on smartphones, tablets or smart devices with metering or remote-control capabilities. The data flows between these two sides of the app are conveyed by the FCM that provides messaging capabilities as topic-based publish/subscribe services. From the client side, the Firebase Messaging API and Android Studio 1.4 or higher with Gradle can be used to design and implement an application to send and receive notifications from FCM belonging to a given set of topics of interest.

When an application wishes to interact with FCM, it needs to obtain a registration token that the FCM SDK generates for the specific application instance running at a given user, and uses this token for authentication. When notifications must be sent from the client application, a proper message must be built containing the information to be sent, jointly with the registration token of the application itself. Then, a proper topic must be indicated in the header of the message, and finally, the message can be sent by using the Admin SDK or the HTTP and XMPP APIs. A topic is a string indicating the subject of the notifications, and is used to discriminate the destinations of the notifications. An application can subscribe to multiple existing topics, or even create new ones, so that if a notification associated with one of these topics is sent to FCM, then the application will receive it back.

FCM delivers topic messages in the same way as other topic-based publish/subscribe services: proper callbacks method must be overridden by the application logic in response to the delivery of a specific kind of notification. FCM has been designed to run in the trusted environment, such as Cloud Functions for Firebase, so as to be reliable (i.e., with 98% of messages delivered to the connected devices), fast (i.e., with a communication latency of 500ms or less), and scalable.

Moreover, it is possible to store the conveyed messages within an instance of the Firebase Realtime Database (i.e., a JSON-based NoSQL cloud-based database), or even a given server-size application with a MySQL database, by using the FCM server APIs and Node.js to dispatch messages and implementing the back-end application logic within the mobile app.

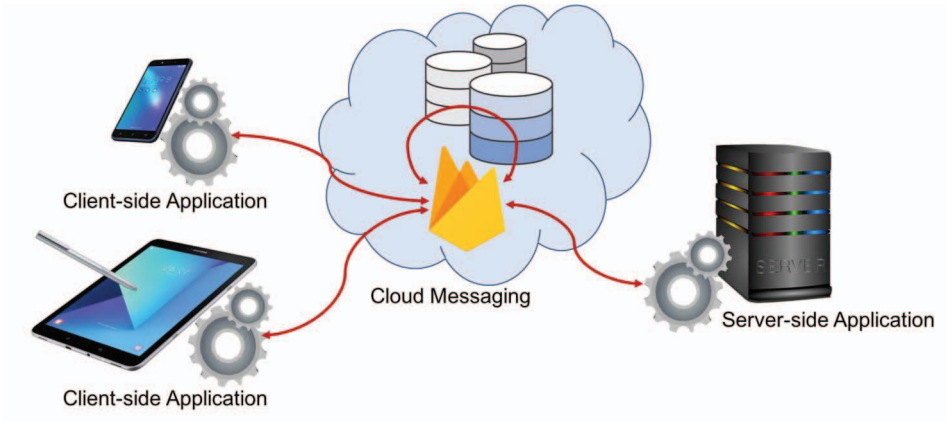

Figure 1. Mobile app architecture with Firebase Cloud Messaging (FCM).

SECURITY AND PRIVACY CONCERNS WITHIN CURRENT CLOUD MESSAGING SOLUTIONS

The use of FCM incredibly eases and speeds up the design and implementation of mobile apps, as well as easing their testing due to the possibility of sending test messages with the Notifications Composer in the Firebase console. However, such cloud-based messaging solutions can pose considerable security and privacy issues that must be adequately considered prior to their adoption in a critical mobile app project.

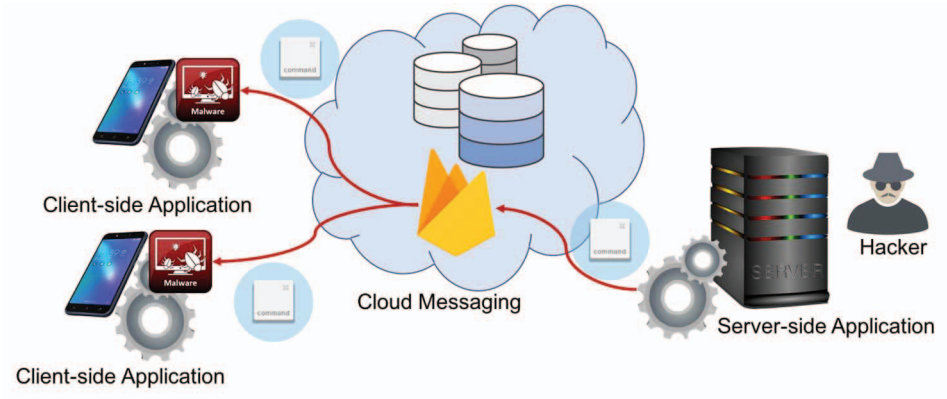

Figure 2. A possible use of a Cloud Messaging solution for the command & control of malware.

The first security issue is related to the possibility of exploiting such solutions by those malicious software agents that fall within the class of badware, as defined within the ENISA threat taxonomy.[7] The predecessor of FCM, called Google Cloud Messaging (GCM), has been reportedly abused to design and realize several malicious advertisement applications. Such applications are widely known as adware, and have the sole intent of displaying ads and consuming a device's battery. Ahmadi and colleagues reported a list of malicious attempts exploiting GCM,[8] where due to applications such as Revmob, Airpush, and Leadbolt, a GCM channel is used to push ads content. However, Zhao and colleagues[9] described how among the many potential malicious uses of a Cloud Messaging service, the most dangerous one is providing the command and

control (C&C) facilities to coordinate a malware infection or a botnet, as Figure 2 shows. Specifically, a botnet that exploits a Cloud Messaging solution as a relay and C&C mechanism can be potentially more powerful, scalable and stealthy than the traditional HTTP, IRC, or SMS-based botnets, and can pave the way to the realization of dangerous, undetectable and silent attacks to mobile devices. Similar attacks have been previously investigated.[10,11]

A concrete example is the most widespread threat detected by Kaspersky Labs in 2013, known as Trojan-SMS.AndroidOS.FakeInst.a. This is a classic SMS Trojan designed to send text messages to the most frequently used contacts in the smartphone's address book, delete incoming messages, generate shortcoming to malicious sites, and/or advertise other malware. Such a Trojan exploits GCM to receive commands, to send messages, to create specific notifications, and/or to receive updates. FCM also can be exploited, for instance the FalseGuide malware has reportedly infected several Android apps by taking advantage of FCM.[12] When installed on a smartphone as part of a legitimate app and used as a Trojan horse, such a malware waits by listening for notifications from FCM, which is used to push malicious software modules to all infected phones that download and run them. The use of a Cloud Messaging facility to convey command messages from a cybercriminal compounds the challenge of tracing the identity and location of the cybercriminal controlling the mobile botnets, due to the decoupling properties provided by such communication solutions. Also, the protection against such threats is not simple, since deactivating the Cloud Messaging services on a given device would potentially render a number of apps installed on the device itself unusable. Furthermore, the only viable solution seems to be blocking only a given notification channel or unsubscribing from a given topic. Existing solutions include using traffic monitoring to detect malicious messages being pushed throughout the Cloud Messaging services.[8] However, it is not straightforward and simple to implement such a kind of solution, since it is necessary to identify in advance the specific channel or topic used by the malicious botnet

Finally, Cloud Messaging may introduce potential privacy issues, by making it possible to obtain the registration ID of the victim's app by hacking GCM or similar services, resulting in sensitive information, such as the victim's Facebook messages, being pushed to the adversary's device.[13] Hence, we need to build a secure end-to-end protection scheme on top of the unsecure communication channels provided by the Cloud Messaging services, and this is why FCM offers payload encryption.[14]

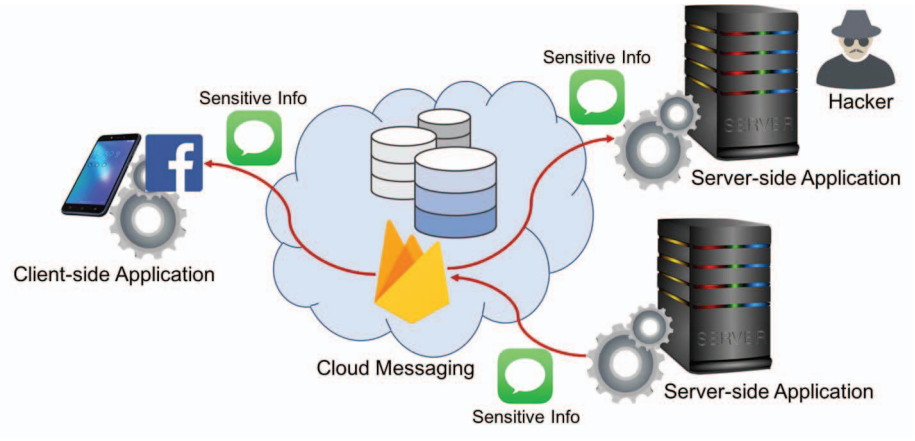

Figure 3. Information leakage by exploiting Cloud Messaging.

BLOCKCHAIN FOR SECURE EVENT-BASED COMMUNICATIONS

Blockchain can be used to secure event-based communications. Such a platform consists of distributed databases that chain together blocks of data by means of cryptographic primitives to make them tamper-proof. In other words, any change to the chain can be assumed as an event to

be notified as the event happens in publish/subscribe services. Such events are made immutable by the blockchain platform so that any further change is a new block in the chain. The difference with publish/subscribe services is that each new block must be validated by a miner with a proof-of-work. As Figure 4 shows, any new block produced by a client application on a smartphone is passed to the blockchain platform hosted in the cloud, which routes it to a miner that validates and approves the new block. Such an approved block is sent back to the cloud to be chained with the other blocks related to the same topic. Any client application can subscribe to a chain so that any changes are notified so as to obtain the new block.

The phase of block validation and approval is crucial, as it facilitates the identification of malicious uses of the platform. In addition, Cloud Messaging provides coarse-gain access control using registration tokens, while the blockchain platforms make use of more fine-grained access control via smart contracts.[15] A first attempt to combine an event-based messaging and blockchain has been proposed by using Hyperledger.[16] Future work in this direction is needed to fully support the event notification functionalities over the blockchain, such as content-based routing or filtering.

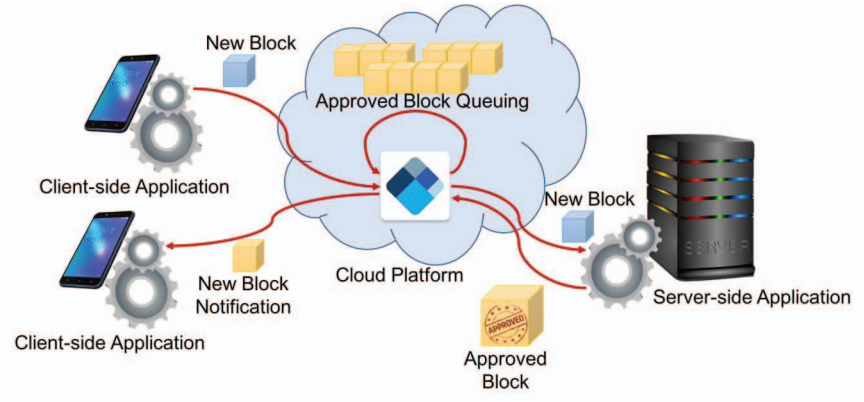

Figure 4. A possible use of the blockchain technology in cloud-based push messaging.

REFERENCES

1. K.-K.R. Choo, "Legal Issues in the Cloud," *IEEE Cloud Computing*, vol. 1, no. 1, 2014, pp. 94–96.
2. M. Cinque, C. Di Martino, and C. Esposito, "On data dissemination for large-scale complex critical infrastructures," *Computer Networks*, vol. 56, no. 4, 2012, pp. 1215–1235.
3. N. Li, Y. Du, and G. Chen, "Survey of Cloud Messaging Push Notification Service," *Proceedings of the International Conference on Information Science and Cloud Computing Companion*, 2013, pp. 273–279.
4. P. Li et al., "Implementation of Cloud Messaging System Based on GCM Service," *Proceedings of the International Conference on Computational and Information Sciences*, 2013, pp. 1509–1512.
5. A. Antonić et al., "A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things," *Future Generation Computer Systems*, vol. 56, no. C, 2017, pp. 607–622.
6. L. Moroney, *The Definitive Guide to Firebase - Build Android Apps on Google's Mobile Platform*, Apress, 2017.
7. L. Marinos, *ENISA Threat Taxonomy - A tool for structuring threat information*, report, European Union Agency For Network And Information Security, 2015; https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/enisa-threat-landscape/etl2015/enisa-threat-taxonomy-a-tool-for-structuring-threat-information.

8. M. Ahmadi et al., "Detecting Misuse of Google Cloud Messaging in Android Badware," *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices* (SPSM 16), 2016, pp. 103–112.
9. S. Zhao et al., "Cloud-based push-styled mobile botnets: a case study of exploiting the cloud to device messaging service," *Proceedings of the 28th Annual Computer Security Applications Conference* (ACSAC 12), 2012, pp. 119–128.
10. M. Ficco and M. Rak, "Intrusion Tolerance of Stealth DoS Attacks to Web Services," *Proceedings of the IFIP International Information Security Conference: Information Security and Privacy Research* (SEC 12), 2012, pp. 579–584.
11. F. Palmieri et al., "Energy-oriented denial of service attacks: an emerging menace for large cloud infrastructures," *The Journal of Supercomputing*, vol. 71, no. 5, 2015, pp. 1620–1641.
12. C. Cimpanu, "Over Two Million Users Infected With New Android Adware," *Bleeping Computer*, 2017; https://www.bleepingcomputer.com/news/security/over-two-million-users-infected-with-new-android-adware/.
13. T. Li et al., "Mayhem in the Push Clouds: Understanding and Mitigating Security Hazards in Mobile Push-Messaging Services," *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security* (CCS 14), 2014, pp. 978–989.
14. M. Scales, "Web Push Payload Encryption," *Google Developers*, 2016; https://developers.google.com/web/updates/2016/03/web-push-encryption.
15. K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, no. 1.1, 2016, pp. 2292–2303.
16. N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: a decentralized, permissioned, publish/subscribe service using blockchains: demo," *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos* (Middleware 17), 2017, pp. 15–16.

ABOUT THE AUTHORS

Christian Esposito is an assistant professor at the University of Naples "Federico II," where he received his PhD in computer engineering and automation. His research interests include reliable and secure communications, middleware, distributed systems, positioning systems, multi-objective optimization, and game theory. Esposito has served as a reviewer or guest editor for several international journals and conferences and has been a PC member or organizer of about 40 international conferences/workshops. He also serves as guest editor for several journals, and is the member of two journal editorial boards. Contact him at christian.esposito@unina.it.

Francesco Palmieri is an associate professor at the University of Salerno, where he received his MS and PhD in computer science. His research interests include advanced networking protocols and architectures and network security. Palmieri has been the director of the Networking Division of the University of Naples "Federico II" and contributed to the development of the Internet in Italy as a senior member of the Technical-Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He serves as the editor-in-chief and editorial board member of several international journals. Contact him at fpalmieri@unisa.it.

Kim-Kwang Raymond Choo holds the Cloud Technology Endowed Professorship in the Department of Information Systems and Cyber Security at the University of Texas at San Antonio. His research interests include cyber and information security and digital forensics. He is a senior member of IEEE, a Fellow of the Australian Computer Society, and has a PhD in information security from Queensland University of Technology, Australia. Contact him at raymond.choo@fulbrightmail.org.

Osmotic Message-Oriented Middleware for the Internet of Things

Thomas Rausch
TU Wien

Schahram Dustdar
TU Wien

Rajiv Ranjan
Newcastle University

Editor: Rajiv Ranjan;
rranjans@gmail.com

Message-oriented middleware is a key technology in today's Internet of Things (IoT). Centralized message brokers facilitate decoupled device-to-device communication and can transparently scale to handle many millions of messages per second. However, Cloud-based solutions, such as AWS IoT or Azure IoT Hub, are challenged to satisfy the stringent Quality of Service (QoS) and privacy requirements of many modern IoT scenarios. Such scenarios are complex because they are not only distributed, but dynamic, as elements physically move, fail, and/or (dis-)connect to/from the network. Instead, distributed middleware needs to leverage the ever- increasing amount of resources at the edge of the network to provide reliable, ultra-low-latency, and privacy-aware message routing. But the heterogeneity and volatility inherent to Edge resources, and the unpredictability of mobile clients, make it extremely challenging to provide resilient coordination mechanisms and guaranteed message delivery. Applying Osmotic Computing principles to message-oriented middleware opens new opportunities for solving these challenges.

Message-oriented middleware (MOM) has undergone several architectural paradigm shifts over the past decades. The scalability issues of using centralized servers for application-layer message dissemination were discussed intensely during the peer-to-peer era, and many solutions based on completely decentralized peer-to-peer architectures were developed.[1] But the forces of monopolization and the widespread adoption of Cloud computing have caused many commercial solutions to return to a centralized approach, where scalability is largely achieved by consolidating resources into massive data centers. Amazon's AWS IoT,[2] or Microsoft's Azure IoT Hub,[3] for example, are massive transparently scaling centralized brokers that can handle billions of messages per second, even in the face of varying load, and at relatively low cost. However, despite

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

further advances in Cloud computing and clustering techniques, the IoT and related application scenarios have introduced new challenges, in particular for centralized systems. IoT applications with ultra-low latency, network resilience, or stringent data privacy requirements cannot be implemented by Cloud-based solutions alone, but demand decentralization. Yet, system engineers have grown accustomed to the convenience and benefits of centralized deployment and management.

Edge computing has been recognized as a solution to this dilemma.[4] By leveraging the ever-increasing amount of resources at the edge of the network for computation, data management, and application-layer message dissemination, many of the QoS and privacy challenges can be addressed. Edge computing also recognizes the Cloud as a necessary and integral part of a holistic system. The Edge can hand off compute intensive tasks to the Cloud, or forward processed data for long-term storage and offline analytics. But how to efficiently and effectively integrate the Cloud and the Edge remains a topic of debate. The heterogeneous nature of edge resources, and their unpredictable availability make it challenging to provision and manage them in a centralized manner. Client mobility further complicates matters, especially with respect to message delivery guarantees and QoS optimization in MOM.

Osmotic computing has been proposed as a new computing paradigm for Edge/Cloud integration.[5] It borrows from the biological principles of osmosis, where solvent molecules seamlessly diffuse into regions of higher solute concentration. Although osmotic computing was originally conceived for the dynamic management of microservices across federated Edge/Cloud infrastructure, it has enjoyed success in other problem areas such as stream processing for IoT or deep learning.[6–7] We argue that osmotic computing principles can be applied to seamlessly integrate Cloud-based MOM into Edge computing. Osmotic computing can abstract how we think about elasticity of MOM towards the Edge, i.e., dynamically moving or provisioning message brokers from the Cloud the Edge based on current demands. In particular, we propose an architecture based on two diffusion models: broker and client diffusion. From a static centralized deployment in the Cloud, we bootstrap a network of brokers that diffuse into the Edge based on resource availability, and the number of clients and their proximity to Edge resources. Clients in close proximity diffuse to the Edge depending on broker proximity and the potential to optimize QoS between the clients. We present an architecture, and proximity-detection and orchestration mechanisms to implement MOM based on osmotic computing principles.

THE NEED FOR EDGE-ENABLED MOM

Modern IoT scenarios highlight the need for MOM that leverages resources at the edge. For example, let us consider the following scenario from the mobile health (mHealth) domain. In a major disaster situation, effective coordination and prompt reaction of emergency teams is paramount to save people's lives. To support on-premises decision-making, first responders attach wearable biosensors to patients, which continuously publish medical signs. Emergency technicians carry mobile devices with software to visualize patient data, trigger alerts, and coordinate team efforts. Cloud-based IoT platforms may be impractical or unreliable in this case, as such health-critical mobile systems must be resilient towards network partitions, and require near-real time data processing and message exchange. Instead, resources in close proximity, such as tactical cloudlets[8] deployed in emergency vehicles must be used to provide necessary services. From there, data can be processed, forwarded to hospitals for further analytics and acute patient care, and handed off to the cloud for long-term storage.

MOM plays a critical role in this type of scenario. It has to facilitate low-latency communication between devices in close proximity, e.g., to provide immediate alerts to changing medical conditions, while still being able to disseminate messages to locations on different levels of geographic dispersion. However, the mobility and unpredictability of both clients and resources that could act as brokers require a completely dynamic operational mechanism. In Cloud-based MOM, operational configuration, such as broker addresses, publish/subscribe topics, etc. are typically best defined at deployment time. In contrast, Edge-based MOM needs to react to spontaneously emerging network structures and device congregations, such as our disaster scenario. This

also means that clients need to be able to quickly locate the closest broker, and QoS optimizations need to consider that proximity between clients and brokers may change during runtime, leading to dynamic routing as end and edge devices change location.

State-of-the-Art

Messaging middleware, publish/subscribe systems in particular, have enjoyed immense research interest over the past decades. Systems like Hermes[9] or PADRES[10] have led to an entire body of research dedicated to optimizing and enabling robust message dissemination in complex peer-to-peer overlay networks. The advent of Cloud computing and the IoT has changed the landscape of real-world messaging solutions dramatically.

Over the past few years, many commercial cloud-based messaging solutions have emerged that target IoT platforms. Amazon's AWS IoT[2], Microsoft's Azure IoT Hub[3] and others all aim at providing transparently scaling IoT device integration via pub/sub messaging. Although these solutions can deal with massive amounts of devices and messages, they cannot satisfy the stringent QoS and latency requirements imposed by many IoT scenarios, as they completely disregard proximity and edge resources.

Some open-source solutions have shown tentative efforts to provide basic mechanisms for enabling real-world edge computing applications. The popular message brokers Mosquitto or HiveMQ, for example, provide the concept of bridging, where brokers can be deployed at the edge and statically configured to forward messages of specific topics to centralized brokers.[11–12] JoramMQ[13] can model a hierarchical broker tree to allow low-latency communication in pre-defined subsystems. These approaches are extremely limited in their operational capability, as they are all static in nature.

Only recently have researchers started to investigate the challenges of IoT and proximity awareness that confronts messaging middleware. MultiPub,[14] for example, manages a broker cluster that spans multiple cloud regions, and optimizes latency for clients based on their proximity to a region. EMMA[15] goes one step further in that it considers brokers on arbitrary edge resources, and dynamically re-configures client-broker connections at runtime. PubSubCoord[16] has also identified Edge/Cloud integration as a challenge for IoT, and proposes a dynamic broker system consisting of edge and routing brokers.

OSMOTIC MESSAGE-ORIENTED MIDDLEWARE

Design Goals of Osmotic MOM

As we have illustrated, the IoT and edge computing architectures provide a great deal of challenges when building messaging middleware. Based on our motivating scenario and existing analyses of edge computing characteristics,[17] we have identified several key design goals that drive the design of our architecture and control mechanisms.

Stringent Latency Requirements

Optimizing end-to-end latencies for clients is one of the key goals of our middleware. Most of the latency in messaging middleware is caused by packet routing, lost packet retransmission processing, and physical propagation delays of network links. Routing messages to the Cloud and then disseminating them from there is wasteful, especially if device-to-device communication happens in close proximity. For these situations, brokers deployed at the edge can provide ultra-low latency communication. At the same time, subscribers may be located at geographically dispersed locations, e.g., a data analytics services in the Cloud, meaning that messages must be forwarded to the Cloud if necessary.

Mobile Brokers and Endpoints

Mobility is a fundamental element of IoT[19] and Edge computing and therefore must be considered as such by osmotic middleware by design. In our scenario, not only clients, but also brokers can be mobile. Proximity between nodes may change any time during runtime, and the system should be equipped with mechanisms to maintain optimized QoS, even in the face of client or broker mobility. Providing message delivery guarantees in the face of mobility is another prominent challenge.

Variable Resource Availability

Resources at the edge are a key component in enabling the previous two goals. However, because these resources are highly dynamic, we cannot predict or rely on their availability. We therefore also cannot provide static operational configurations. Instead, a central configured deployment should be able to organically diffuse into edge regions where and when necessary. A key mechanism required to facilitate this is proximity detection.

Monitoring and Management

Looking at the previous goals, it becomes clear that proximity is a fundamental concept in IoT and edge computing, and needs to become a first-class-citizen not only in messaging middleware, but edge computing algorithms and systems in general. This would appear to imply that location of brokers and endpoints must be determined in real-time and their physical relationship to each other must be deduced in terms of the local topology. As we'll see, proximity can instead be determined by calculating effective network latency. Additionally, routing of messages between elements must balance reliability and performance considerations. Moreover, scalability of proximity monitoring is a key challenge to avoid excessive strain on the network and resource constrained devices.

IoT Heterogeneity

A plethora of messaging protocols and client hardware for the IoT exists that may be difficult to replace or update.[18] Existing infrastructure should seamlessly integrate with a distributed messaging middleware. Furthermore, middleware control mechanisms need to be completely transparent to end devices.

Architecture

Figure 1 shows a high-level overview of our architecture, which includes the components that follow.

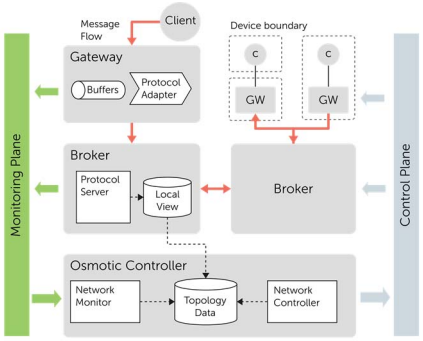

Figure 1. The architectural components of osmotic message-oriented middleware. The osmotic controller orchestrates a network of brokers and gateways via a monitoring and control plane. The gateways transparently connect clients to the system.

Broker Network

Brokers implement a server-side messaging protocol (e.g., MQTT), and facilitate message dissemination to clients and within the broker network. Because brokers are treated as an ephemeral resource, their control mechanisms have to be entirely dynamic. This particularly concerns addressing and message routing. However, maintaining a global view of the network topology at each broker would be impractical. Instead, the Cloud maintains a global view, whereas brokers only maintain a local view of the topology and routing tables necessary to operate within their contextual boundary. Brokers also monitor and provide access to key performance indicators, such as resource consumption, message throughput, average routing delay, etc., to provide coordination and load balancing algorithms with necessary data.

Gateway Network

Gateways are an integral part of the coordination fabric and provide clients transparent access to the broker network. They serve two main purposes: 1) to seamlessly integrate existing IoT infrastructure into the network, and 2) to enable client and broker mobility. Gateways act as intermediaries that intercept protocol control packets from clients to understand their context (e.g., connection information, Message Delivery Agreements (MDAs) and topic subscriptions) and allow the transparent reconnection of clients to other brokers when adapting to varying network QoS or topology changes. They serve as proxies to measure proximity between clients and brokers. Gateways also do simple data processing such as filtering or protocol translation via protocol adapters. Because they are lightweight, gateways can typically be easily hosted by resource-constrained IoT devices, and can serve either one or multiple clients.

Osmotic Controller

The controller maintains a full view of the network topology and is responsible for bootstrapping and orchestrating the system. New brokers and gateways register with the controller when they enter the network, and continuously report data to the controller via the monitoring plane. The controller acts as a registry and discovery service for edge resources to which brokers could potentially be provisioned. Brokers are not dependent on the controller to process and forward messages because they maintain a local view of the network topology and routing tables. However, the osmotic controller is required to 1) diffuse clients, i.e., enacting QoS optimization decisions by instructing gateways to connect to other brokers, and 2) diffusing brokers, i.e., elastic provisioning of brokers to edge resources.

Protocol Adapters

Protocol adapters are a subcomponent of gateways and a fundamental part in controlling the heterogeneity inherent to IoT. With these adapters, gateways can act as protocol bridges. For example, clients may send messages via a different protocol than implemented by the middleware. To seamlessly integrate these clients into the system, a protocol adapter on the gateway translates messages from the client protocol to the server protocol. Because of the plethora of protocols available in the IoT,[17] it would be impractical to equip every gateway with all existing adapters at deployment time. Instead, gateways can pull adapters from the controller on demand at runtime based on their context and connected clients.

Monitoring Plane

Monitoring a network's topology and QoS is a key element for edge-enabled message-oriented middleware. It is particularly important for determining proximity between clients and brokers, as proximity is calculated from network metrics such as latency, routing hops, or physical measurements such as signal strength. Both gateways and brokers need to be able to measure and report on their surrounding network state. We discuss proximity detection, and QoS and demand monitoring in more detail in Elastic Diffusion.

Control Plane

A main function of the osmotic controller is to provision brokers to edge resources, and coordinate client diffusion. To that end, the controller issues control commands via the control plane to brokers and gateways. Brokers and gateways provide control endpoints that accept coordination commands, such as an instruction to reconnect to a different broker. We discuss provisioning and network reconfiguration in more detail in Elastic Diffusion.

Elastic Diffusion

As we have established, Cloud-based message brokering cannot satisfy the stringent QoS requirements of IoT applications. This particularly concerns device-to-device communication in close proximity. Instead, brokers at the edge should facilitate low-latency communication for devices in close proximity, and forward messages to the cloud for further dissemination. This already works well for simple configurations and static infrastructure. For example, Mosquitto can be used to statically forward specific topics from one broker to another. However, as our motivational scenario illustrated, edge computing applications require a dynamic system that can adapt to changing network QoS and topology, while still allowing centralized configuration and management. During runtime, when clients at the edge require low-latency communication, the broker network needs to be expanded by provisioning brokers to edge resources dynamically. The routing topology needs to be updated, and clients in close proximity need to be migrated to the new broker. When client communication stops, these brokers can be discarded, and the network may shrink again. We call this process elastic diffusion. Figure 2 illustrates the concept.

Starting from a single configuration in the Cloud, i.e., an osmotic controller and a root broker, the system grows and shrinks the broker network during runtime to adapt to changing network topology, client mobility and demand, and resource availability. Brokers diffuse to edge resources when there is potential to optimize client QoS. Similarly, clients diffuse to edge brokers to reduce end-to-end latencies, and to balance load between brokers. A key mechanism to facilitate this is a concept we call osmotic pressure.

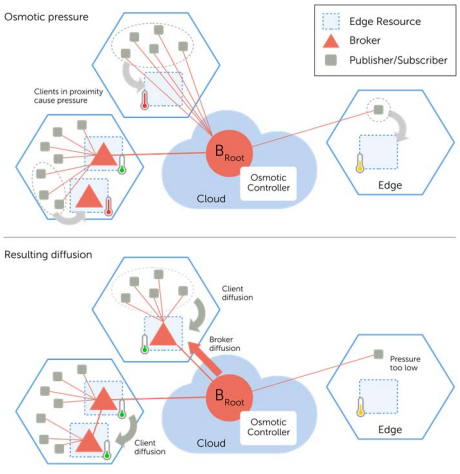

Figure 2. Osmotic pressure facilitates controlled diffusion of brokers and clients to the edge. Clients exert osmotic pressure on local resources depending on their proximity and demand. The osmotic controller aims to balance osmotic pressure throughout the system.

Osmotic Pressure

To facilitate elastic diffusion, we first need an aggregated quantification of proximity and demand. To that end, we define a metric inspired by a core concept of osmosis: osmotic pressure. Similar to the biological process, in our system, osmotic pressure causes brokers to diffuse onto

edge resources. Clients in close proximity to edge resources cause a "pulling" effect on those resources. When the pressure exceeds or drops below a given threshold, the system deploys or removes brokers from these resources. Similarly, brokers deployed on edge resource cause a "pulling" effect on clients in close proximity, which causes a reconfiguration of client-broker connections to optimize QoS for those clients. For edge resources, we calculate osmotic pressure based on the amount of clients and their proximity to the resources. Osmotic pressure is high when many clients exist in close proximity to a resource. The exerted pressure is increased with the message throughput of these clients, i.e., if local demand is high.

Proximity Detection & Demand Monitoring

To effectively calculate osmotic pressure, we need 1) a good quantification of the proximity between brokers and resources, 2) a scalable way to continuously monitor this proximity, and 3) a way to monitor demand of clients. In other distributed systems, such as Content Distribution Networks (CDN), closeness between nodes is often defined and determined by geo location, routing hops, or round-trip times (RTT). For 1), we argue that, because the main goal is to optimize end-to-end latency between clients, using the effective network latency is the best way of determining proximity between clients and resources. Additionally, for determining proximity between clients and brokers, brokers can add their average message buffering delay to the effective network latency to calculate a more precise RTT value. After all, a "close" broker on a congested resource can behave as if it were physically much farther away. However, in contrast to CDN, where proximity is determined on a per-request basis via DNS probing, osmotic MOM needs to continuously monitor the effective network latency between all nodes, which introduces serious scalability challenges. Simple monitoring solutions can cause high network overhead that becomes difficult to manage[15] For 2), we therefore propose a synthesis of network location approaches[19] with concepts such as interest management, where the frequency of monitoring is concentrated to relevant regions. For 3), a combination of throughput monitored from brokers and gateways can be used to estimate demand for a specific region, topic or content type.

Broker Diffusion

Like in osmosis, where solute molecules exert osmotic pressure, clients in close proximity of edge resources exert osmotic pressure on these resources, causing an imbalance in the network. To balance the osmotic pressure, the osmotic controller reacts by deploying a broker to the resource. We assume that edge resources provide some form of container-based virtualization, e.g., Docker, which allows the controller to deploy new brokers as containers. The new broker is integrated into the network and the controller reconfigures the routing overlay, i.e., dynamically creates topic bridges to forward messages to the cloud or other brokers if necessary. When the osmotic pressure drops, e.g., because clients move to other locations or leave the network, brokers may be discarded by the system.

Client Diffusion

Clients' connections are migrated into edge regions based on their proximity to brokers deployed at the edge. If brokers are deployed at the edge due to osmotic pressure, clients will diffuse there. Because osmotic pressure is calculated using both proximity and broker performance indicators, we have a twofold effect: clients will experience better end-to-end latencies as link usage is minimized, and load is balanced between brokers that provide similar QoS to those clients. When brokers are discarded because of low osmotic pressure, or because of a broker outage, clients diffuse back to a broker closer to the cloud.

Message Delivery Agreement

When it comes to message delivery guarantees, there are always trade-offs between the level of consistency and the end-to-end latency a system can provide, between transmission reliability and bandwidth utilization, and between transmission delay and processing requirements and se-

quencing. We recognize that even a single application can have several types of consistency demands. Take for example an analytics tool in our mHealth application. For some subscribers, it may be critical to receive every single data point in the correct order and without duplicates. For others, duplicate data may not be a problem, but the requirement is ultra-low latency delivery. The MQTT protocol addresses this in part via its "quality of service" concept, which defines the type of delivery guarantee a broker provides: at least once, at most once, and exactly once delivery.

In a single-broker system, implementing these types of guarantees is relatively straight forward, as MQTT demonstrates. However, in our distributed broker system, the type of consistency that the system must provide significantly affects how brokers and clients have to be coordinated during diffusion. For example, when a subscriber to a specific topic is migrated from one broker to another, messages received by the target broker during the reconnection process may not reach the migrating client. This may be problematic for some applications, but a coordination mechanism that provides this type of consistency guarantees can be costly in terms of resource consumption and latency penalties.

To achieve more granular control over this, we propose the concept of MDA that are negotiated between clients and brokers. An MDA is a type of Service Level Agreement (SLA) regarding message delivery. It controls how much effort the broker network makes to deliver messages between clients for a given type of topic or content, and provides the client with an estimate of the incurring latency penalty.

MDAs have two parameters: a consistency guarantee and a latency estimate. The consistency guarantee is closely related to MQTT's message delivery QoS: i.e., defines an "at least once", "at most once", or "exactly once" delivery semantic. The latency estimate is calculated from the current state of the network, the current broker workload, and the latency penalties caused by providing increased consistency. When a client requests a consistency guarantee for a specific type of topic or content, the broker responds with whether it can provide this guarantee, and if so, an estimate of how long message dissemination will take under this type of guarantee. This mechanism helps fine-tune an application's response time for clients, allows clients to handle the trade-off between latency and consistency, and allows the system to optimize coordination. When highly consistent delivery is required, the system must make extra efforts to maintain consistency during broker or client diffusion.

CONCLUSION

How to best achieve seamless Edge/Cloud integration to fully enable the Internet of Things is still subject of debate. The increasing interest in Edge computing has led to new computing paradigms that disrupt current approaches for IoT systems. One such paradigm is osmotic computing. We have shown how principles of osmotic computing can be applied to message-oriented middleware, to provide a distributed network of brokers that elastically diffuse to edge resources on demand. Unlike Cloud-based IoT platforms that typically completely neglect the proximity of devices, osmotic MOM elevates proximity to be of primary concern, both for deploying new brokers, and to optimize responsiveness for clients. Our architecture can serve as stand-alone messaging middleware to facilitate device-to-device communication in IoT environments, and can also complement complex edge computing applications that rely on message brokers, such as distributed real-time data analytics applications.

REFERENCES

1. P.T. Eugster et al., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, 2003.
2. *AWS IoT*, Amazon; azure.microsoft.com/en-us/services/iot-hub.
3. *Azure IoT Hub*, Microsoft; azure.microsoft.com/en-us/services/iot-hub.
4. W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, no. 5, 2016, pp. 78–81.

5. M. Villari et al., "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," *IEEE Cloud Computing*, vol. 3, no. 6, 2016, pp. 76–83.
6. M. Nardelli et al., "Osmotic Flow: Osmotic Computing + IoT Workflow," *IEEE Cloud Computing*, vol. 4, no. 2, 2017, pp. 4–2.
7. A. Morshed et al., "Deep Osmosis: Holistic Distributed Deep Learning in Osmotic Computing," *IEEE Cloud Computing*, vol. 4, no. 6, 2017, pp. 22–32.
8. G. Lewis et al., "Tactical Cloudlets: Moving Cloud Computing to the Edge," *Proceedings of the 2014 IEEE Military Communications Conference*, 2014, pp. 1440–1446.
9. P.R. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," *Proceedings of the 22nd International Conference on Distributed Computing Systems* (ICDCSW 02), 2002, pp. 611–618.
10. E. Fidler et al., "The PADRES Distributed Publish/Subscribe System," *Feature Interactions in Telecommunications and Software*, 2005, pp. 12–30.
11. M. Garcia, "How to Bridge Mosquitto MQTT Broker to AWS IoT," *The Internet of Things on AWS -- Official Blog*, blog, 2016; https://aws.amazon.com/blogs/iot/how-to-bridge-mosquitto-mqtt-broker-to-aws-iot.
12. *HiveMQ*; www.hivemq.com.
13. *JoramMQ, a distributed MQTT broker for the Internet of Things*, white paper, ScalAgent, 2014.
14. J. Gascon-Samson, J. Kienzle, and B. Kemme, "MultiPub: Latency and Cost-Aware Global-Scale Cloud Publish/Subscribe," *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems* (ICDCS 17), 2017, pp. 2075–2082.
15. T. Rausch, S. Nastic, and S. Dustdar, "EMMA: Distributed QoS-Aware MQTT Middleware for Edge Computing Applications," *IEEE International Conference on Cloud Engineering*, 2018.
16. K. An et al., "An Autonomous and Dynamic Coordination and Discovery Service for Wide-Area Peer-to-peer Publish/Subscribe: Experience Paper," *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, 2017, pp. 239–248.
17. J. Weinman, "The 10 Laws of Fogonomics," *IEEE Cloud Computing*, vol. 4, no. 6, 2017, pp. 8–14.
18. A. Al-Fuqaha et al., "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, 2015, pp. 2347–2376.
19. B. Wong, A. Slivkins, and E.G. Sirer, "Meridian: A Lightweight Network Location Service Without Virtual Coordinates," *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications* (ACM SIGCOMM), vol. 35, no. 4, 2005, p. 85.

ABOUT THE AUTHORS

Thomas Rausch is a PhD student and Research Assistant at the Distributed Systems Group at TU Wien, Austria. His research interests include Internet of Things, Edge computing, and event-based systems. Rausch has a master's in Software Engineering & Internet Computing from TU Wien. Contact him at t.rausch@dsg.tuwien.ac.at or dsg.tuwien.ac.at.

Schahram Dustdar is a full professor of computer science heading the Distributed Systems Group at TU Wien, Austria. His work focuses on Internet technologies. He is an IEEE Fellow, a member of the Academy Europeana, and an ACM Distinguished Scientist. Contact him at dustdar@dsg.tuwien.ac.at or dsg.tuwien.ac.at.

Rajiv Ranjan is a reader in the School of Computing Science at Newcastle University, UK; chair professor in the School of Computer, Chinese University of Geosciences, Wuhan, China; and a visiting scientist at Data61, CSIRO, Australia. His research interests include grid computing, peer-to-peer networks, cloud computing, Internet of Things, and big data analytics. Ranjan has a PhD in computer science and software engineering from the University of Melbourne. Contact him at raj.ranjan@ncl.ac.uk or http://rajivranjan.net.

Cloud Automation and Economic Efficiency

Eric Bauer
Nokia

Editor:
Joe Weinman,
joeweinman@gmail.com

Understanding the mechanisms by which cloud automation produces efficiency improvement lets organizations appropriately plan their investments in such automation—along with quality and performance improvements—to optimize operational efficiency.

This column offers conceptual and cost models of cloud automation from a longer monograph I've written to help cloud users and service providers understand the feasible and likely mechanisms to effectively drive continuous efficiency improvement.[1] Although cloud computing, and therefore cloud automation, also can generate strategic and revenue benefits, this column will focus on cost savings mechanisms arising from cloud automation.

AUTOMATION BASICS

A common definition of *automation* is "automatically controlled operation of an apparatus, process, or system by mechanical or electronic devices that take the place of human labor."[2] Such control loops predate the age of automation: for example, military strategist John Boyd defined the "OODA" loop: Observe, Orient, Decide, Act. Figure 1 gives a basic model of automation derived from a classic industry analysis,[3] with automated driving as the example:

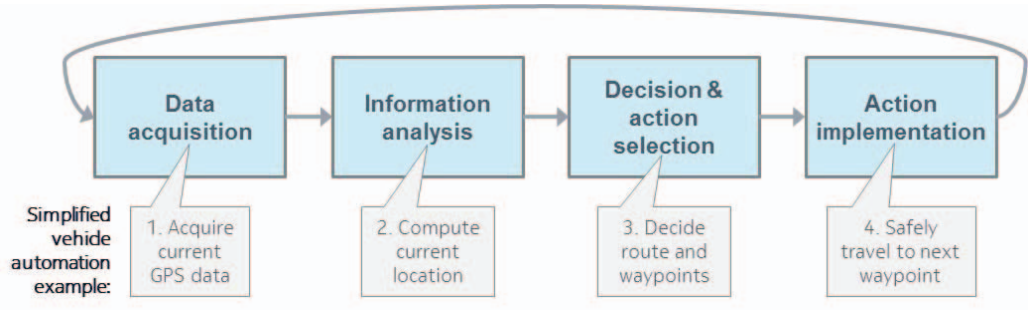

Figure 1. Basic model of automation: Automated driving example.

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

1. **Data acquisition**: Automated mechanisms capture data on current operational status, context, and more. For example, an automated vehicle acquires GPS data as well as various sensor inputs, such as wheel rotation speed, direction, location of other vehicles and so on.
2. **Information analysis**: Automated mechanisms analyze the data to produce information on current state (e.g., vehicle location and velocity), operating context, and more.
3. **Decision and action selection**: Deciding what action is appropriate to take. For example, an automated vehicle must decide on the sequence of waypoints to follow from the current location to the desired destination, and avoid collisions with pedestrians, vehicles and other objects.
4. **Action implementation**: Executing the selected action. For example, an automated vehicle must safely travel from its current location to the next waypoint, which entails braking, acceleration, and steering.

This basic model of automation applies to hybrid or partial automation arrangements as well. For example, Uber today automatically computes the route and waypoints for each vehicle (i.e., automated high-level decision & action selection), but relies on human drivers to safely travel from one waypoint to another. Fully automated Uber vehicles would presumably automate all action implementation to safely travel the selected route without human assistance.

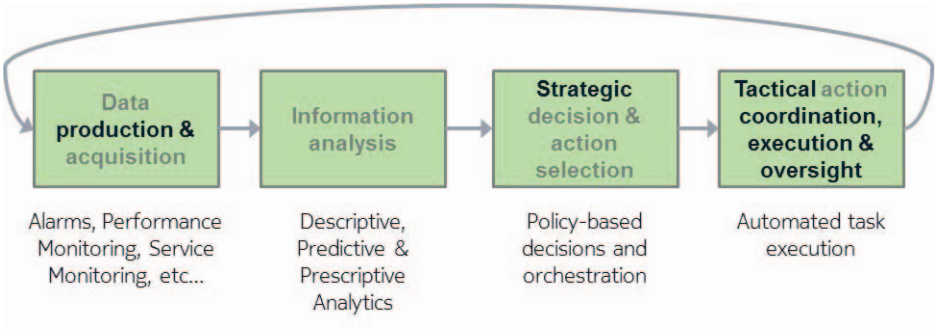

Figure 2. Basic model of cloud automation.

Figure 2 adapts the basic automation model of Figure 1 to operation of a cloud-based application:

- **Data production & acquisition**: Virtualized and physical network elements, as well as functional components offered as a service (a.k.a., PaaS) and service probes, automatically generate myriad alarm, performance, and other data. Customer care and self-service activities also produce important service data, such as user-generated problem reports.
- **Information analysis**: Descriptive and predictive analytics are performed on acquired data to deduce information like the true operational status of the service and all included components, and localize the cause of suboptimal service quality or performance.
- **Strategic decision and action selection**: Policy-based decisions are made and specific actions are ordered by automated orchestration systems (or skilled human), such as diagnosing exactly which component has failed and ordering appropriate repair action.
- **Tactical action coordination, execution and oversight**: Management systems (or skilled humans) coordinate, execute and verify successful completion of an ordered action, like repairing a failed component or elastically scaling an application instance.

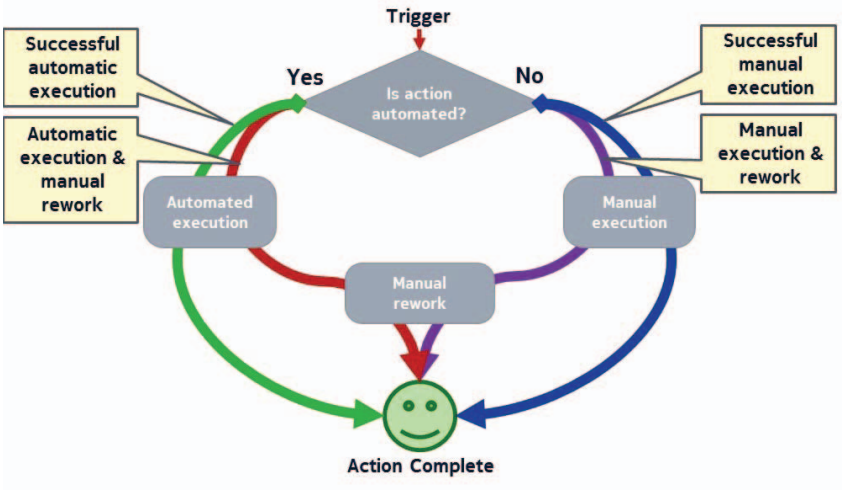

Figure 3. Simple action model.[1]

Tactical action is usefully understood via the simple model of Figure 3. Action is triggered by some strategic decision and action selection. Actions that are automated and execute right the first time promptly traverse the left-most green arc to successful completion. Automated actions that are not right-first-time often require costly and time-consuming, and often manual, rework before they are successfully completed. Manually executed actions generally take longer and cost more than automated actions; manual actions that are not right-first-time take even longer and cost even more.

COST MODELING CLOUD OPERATIONS

Both the key cloud characteristics of on-demand self-service and rapid elasticity and scalability leverage automation and interlock with measured service to "offer [organizations] value by enabling a switch from a low efficiency and asset utilization business model to a high efficiency one."[4] Combining the cloud automation model of Figure 2 and the simple action model of Figure 3 with the activity-based cost model described in my previous work[1] highlights key aspects of this shift to a high efficiency operating model. Figure 4 maps both the cloud automation and simple action models of Figure 3 onto the activity-based cost model of Economic Efficiency of Cloud-Based Application Services.[1]

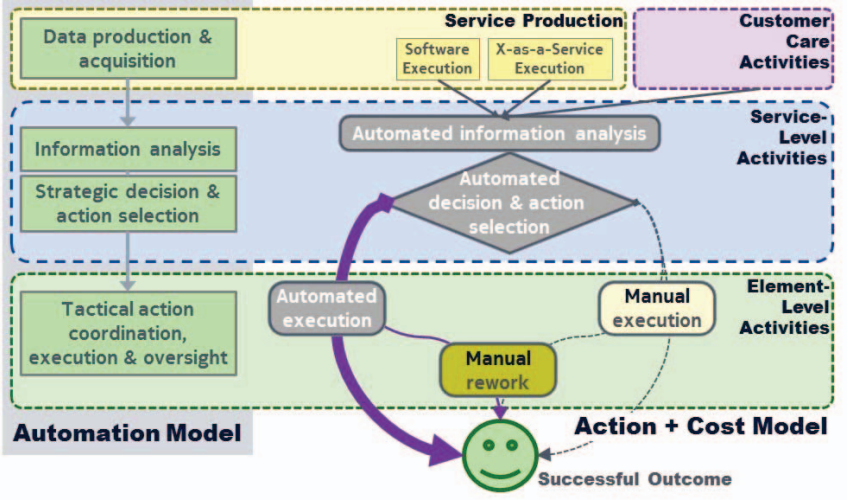

Figure 4. Cost modeling automated operation.

- **Service production and customer care** cost modeling activities cover *data production and acquisition* – direct execution of application components running on virtualized or physical infrastructure, functional components offered as a service and user request fulfillment and problem management activities produce alarms, service quality indicators, performance monitoring and myriad other data that are pushed to information analysis systems. Data production and acquisition is entirely automated for cloud-based application services; after all, there aren't likely to be humans wandering around with clipboards observing service production and recording data on forms. Customer care activities also generate relevant data, such as end user trouble tickets. Costs of producing and acquiring this raw data are modeled under service production and customer care activities.
- **Service-level activities** from the cost model cover:

    1. **Information analysis**: Raw data must be analyzed to produce usable information, such as detecting service impairments, inferring trends and correlating faults. Raw data is sometimes noisy, incomplete or otherwise imperfect, so data scrubbing may be necessary. Typically, all routine information analysis is automated, but humans may perform some data scrubbing and advanced analysis. Note that a nugget of usable information may drive several distinct activities; for example, a detected service impairment might trigger both maintenance actions to localize and correct the root cause of the impairment, as well as capacity management actions to mitigate the service impact of the event. Thus, organizations may aggregate all data storage and information analysis costs into a single (service-level activity) cost object, and perhaps allocate some or all those costs to the activities that benefit from the information.
    2. **Strategic decision and action selection**: The organization's operational policies and practices are applied to the output of information analysis to decide if, when and what actions are appropriate to execute. In most operational domains, the strategic decision & action selection can be aggressively automated.

- **Element-level activities** from the cost model cover *tactical action coordination, execution & oversight*: Automated mechanisms coordinate and synchronize task execution of various infrastructure, platform and software component actions and verify successful completion. Human action is required for non-automated and faulty actions. Costs to coordinate, execute & oversee tactical actions, as well as costs to rework failed actions and the costs of non-productive resources consumed by work-in-process (WIP) are captured as element-level activities.

MECHANISMS OF EFFICIENCY IMPROVEMENT

Figure 5 highlights the primary mechanisms by which automation drains cost from application operation.

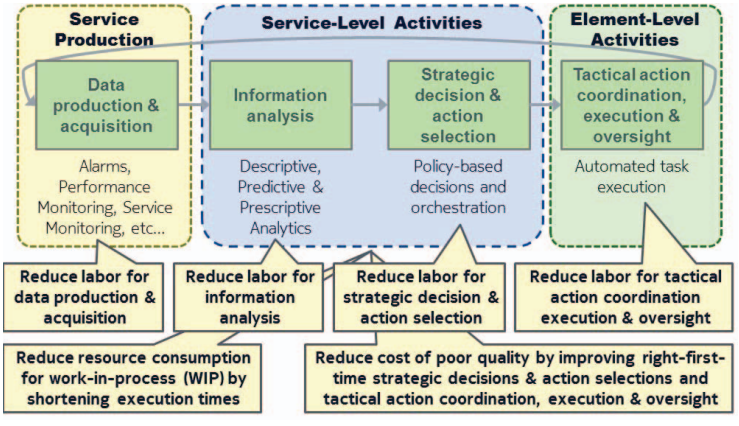

Figure 5. Efficiency improvement mechanisms of automation theme.

1. Reduce human labor data production & acquisition activities by eliminating staff effort from capturing, communicating and storing all service- and component-related data.
2. Reduce human labor for data cleansing and information analysis activities by adopting advanced analytics systems.
3. Reduce human labor for strategic decision & action selection by shifting effort to advanced and policy-based orchestration systems.
4. Reduce human labor from tactical action coordination, execution & oversight via advanced task automation mechanisms.
5. Reducing resources consumed by work-in-process (WIP, especially resources held before or after they are available to serve user demand) by shortening execution times. When organizations pay for resources based on the time the resource is held, reducing the time nonproductive resources are held reduces costs. For example, virtual compute resources are allocated at the beginning of a component instantiation action, but those compute resources are not available to deliver valuable user service until the component instantiation action has completed successfully; if the component instantiation action can be streamlined and the duration of non-productive time that resources are held minimized, then the organization's resource costs can be reduced.
6. Reducing costs of poor quality, especially direct costs of rework as well as liquidated damage remedies for violating quality commitments and soft costs of poor user service quality. Automated execution is inherently more reliable than manual execution thereby producing higher right-first-time execution quality, so less resource intensive rework should be required.

OPTIMAL SCALABLE CAPACITY MANAGEMENT

The rapid elasticity and scalability key characteristic of cloud computing[4] enables online application capacity to track with varying user demand to reduce wasteful consumption of cloud resources like virtual machine instances. Online application capacity beyond what is required to serve actual user demand with acceptable quality and business risk is wasteful, so organizations can improve their operational efficiency by having real-time application capacity track closer to the target online capacity necessary to serve actual demand with acceptable quality and risk.[5]

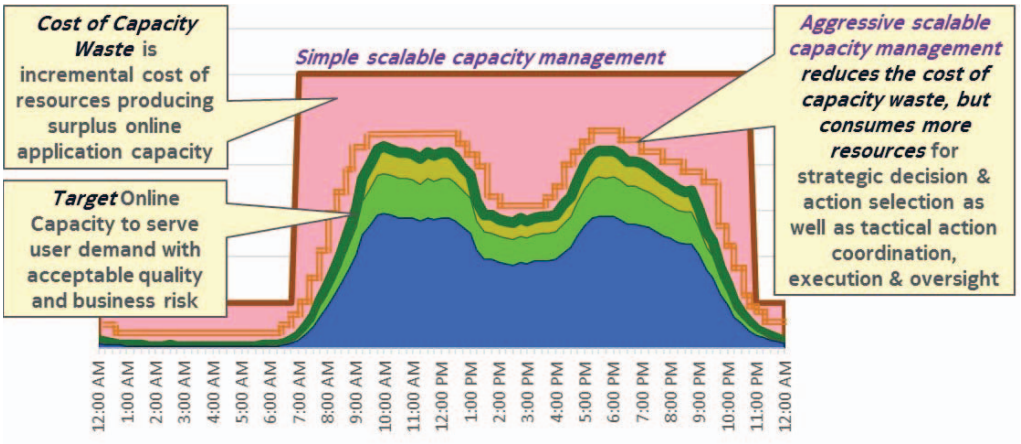

Figure 6. Simple versus aggressive scalable capacity management.

The pink region in Figure 6 between the simple scalable capacity management line and the green target online capacity curve represents surplus online application capacity. Driving the scalable capacity management behavior closer to target online capacity via aggressive scalable capacity management reduces resources consumed by wasteful service production without unacceptably increasing business risk or compromising quality. Consider the automated scalable capacity management models visualized in Figure 6:

- **Simple scalable capacity management**: Executes two capacity changes per day to deploy high online capacity during peak hours (e.g., 7am to 11pm) and low online capacity during off-peak hours. Automating two simple capacity changes per day has modest needs for data production & acquisition; information analysis; strategic decision & action selection; and tactical action execution coordination, execution & oversight so capacity management costs are modest. However, substantial resource capacity is wasted which increases the organizations operating expenses.
- **Aggressive scalable capacity management**: Executes dozens of capacity changes per day to closely track the target online capacity necessary to serve user demand with acceptable quality and business risk. Real-time scaling of online application capacity to follow actual demand requires more real-time data, richer information analysis, more frequent and sophisticated strategic decision & action selection, and more frequent tactical coordination, execution and oversight of capacity change actions; thus, aggressive scalable capacity management is more costly than simple scalable capacity management. However, aggressive capacity management significantly reduces the resource consumption wasted by surplus online application capacity.

As Figure 7 shows, optimal scalable capacity management balances an organization's incremental cost capacity waste savings from eliminating surplus resource consumption against the incremental costs of more aggressive scalable capacity management. For example, more aggressive scalable capacity management models that gather more data, evaluate capacity decisions more frequently and ultimately execute more capacity changes per day have higher costs than simpler capacity management models. Methodically modeling both the cost of scalable capacity management and the cost of capacity wastage for that scalable capacity management model enables one to methodically analyze and optimize the various architectural characteristics and operational policies to improve business results.

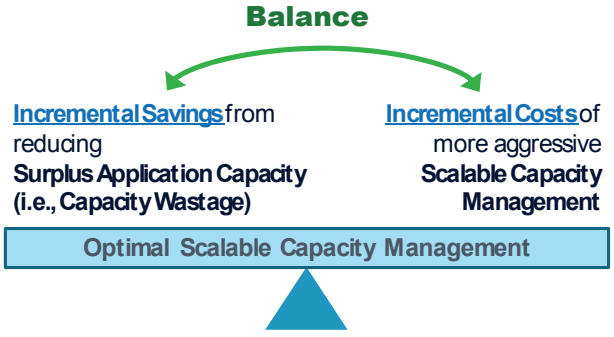

Figure 1. Economically optimal scalable capacity.

CONCLUSION

Organizations accrue cost savings from automating operations of cloud-based application via:

- Reduced labor input to acquire, analyze, make decisions and take service operations actions
- Improving right-first-time execution quality of service operations actions which reduces the intensity of costly rework activities
- Reducing the time non-productive resources are held as work-in-process due to faster action execution

Modeling can quantitatively estimate the costs of various automated operations scenario for cloud-based applications to help organizations predict the feasible and likely efficiency benefits of potential investments in automation, quality and performance improvement and changes to operational policies.

Read *Economic Efficiency of Cloud-Based Application Services* for further information.[1]

REFERENCES

1. E. Bauer, *Economic Efficiency of Cloud-Based Application Services*, 2017; doi.org/http://dx.doi.org/10.13140/RG.2.2.22188.56969.
2. *Dictionary*, Meriam-Webster, 2017; https://www.merriam-webster.com/dictionary/automation.
3. R. Parasuraman, T.B. Sheridan, and C.D. Wickens, "A Model for Types and Levels of Human Interaction with Automation," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 30, no. 3, 2000, pp. 286–297.
4. *Information technology -- Cloud computing -- Overview and vocabulary*, standard ISO/IEC 17788:2014, ISO/IEC Information Technology Task Force, 2014.
5. E. Bauer, "Improving Operational Efficiency of Applications via Cloud Computing," *IEEE Cloud Computing*, vol. 5, no. 1, 2018, pp. 12–19.

ABOUT THE AUTHOR

Eric Bauer is a Bell Labs Fellow and author of six Wiley-IEEE Press books, including *Reliability and Availability of Cloud Computing* and *Lean Computing for the Cloud.* His research interests include service quality, reliability and efficiency of cloud-based application services. Bauer received an MS in Electrical Engineering from Purdue University. He has been awarded 24 US patents. Contact him at Eric.Bauer@nokia.com.

Approaching Cloud Computing Performance

David S. Linthicum
Deloitte Consulting

Performance in a cloud computing environment is a complex topic. The way that IaaS and PaaS clouds operate is very different from traditional single-tenant on-premises platforms. Therefore, the way you evaluate and test IaaS and PaaS clouds is different as well.

Teradata Universe EMEA 2018 recently stated: "A majority of the largest companies in the world (83 percent) agree that the cloud is the best place to run analytics, according to a new survey by Vanson Bourne on behalf of Teradata (NYSE: TDC), the leading cloud-based data and analytics company. In the next five years, by the year 2023, most organizations want to run all of their analytics in the cloud. But, an overwhelming 91 percent say that analytics should be moving to the public cloud at a faster rate."[1]

So, what's wrong with the survey numbers? "According to the survey, some of the biggest barriers to moving analytics to the cloud are security (50 percent), *immature and low-performing available technology (49 percent),*[emphasis added] regulatory compliance (35 percent) and lack of trust (32 percent)." [1]

While performance is typically included with a list of other concerns, such as security, privacy, governance, and compliance, it's coming up more and more as enterprises move to the public cloud with mixed results.

Most enterprises that leverage public cloud platforms believe that elasticity and performance come along for the ride. However, as systems get larger and require more resources, it has become obvious that public clouds have specific performance bottlenecks that must be understood and dealt with.

As more benchmarks are published, we'll see more instances where public cloud providers have systemic performance issues that are not easy to fix. While many will toss faster hardware at the problem, the culprit is typically the way the cloud is designed. Or, more often, how the workloads themselves are designed and deployed.

EMERGING CASE STUDIES

The approaches to dealing with tenant management vary greatly from cloud-to-cloud, and have different performance and scaling characteristics. Also, some cloud providers have widely distributed datacenters to reduce the number of hops it takes to get to them, and thus reduce latency.

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD

Other providers are more centralized and may have latency issues for clients who are geographically far away from the datacenter.

Performance issues, such as the latency problems, can only truly be solved with a complete public cloud restoration project, which is unlikely to happen. That means we'll probably see a few of the public cloud providers faceplant due to a systemic performance problem that they just can't solve in time to capture the market. This is the same pattern repeating itself that appeared within other emerging technologies over the years. Cloud technology is just the latest repeat.

So, what are those in enterprise IT to do when considering cloud-computing performance? There are some basic performance concepts that you should consider as you select public cloud providers, and as you migrate data and processes to the cloud.

It's interesting that almost 50 percent of applications that migrate to the cloud are reported as performing worse or the same when on a better, more "muscled up" public cloud platform. The amount of memory you select or the number of cores that are part of your configuration have little to very random effects. This, according to my experience as a practitioner in the space for the last 10 years.

These performance issues may seem to randomly crop up, but we're actually seeing a few patterns start to emerge:

- **Failure to leverage cloud-native features.** Applications try to find their best path through the platforms hosted in the cloud. In many cases, what the application requests and what the cloud platform provides don't match up, in terms of optimizing performance. For instance, I/O systems have more layers on the cloud platform, and thus the response will lag the on-premises I/O system.
- **Unwillingness to make applications cloud-native.** Most enterprises opt for the lift-and-shift option when it comes time for application migration. It's cheap, fast, and low risk. However, the end results could be less than desirable, such as lower performing application workloads, or higher running costs in the cloud because an application wasn't optimized to leverage public cloud resources.

There are other macro pattern problems to remember as well, such as distance between servers and consumers, poor application design, and the public cloud and its approach to tenant management.

NETWORK LATENCY

Generally speaking, the distance between you and your cloud center will determine the amount of latency you'll experience. You should select a provider with a regional data center that's fairly close to the people or systems that will consume the cloud service. You should test the connection as well, understanding that shorter distances don't always guarantee better network response.

Most public cloud computing providers allow you to lock systems into regional datacenters. That means the providers typically won't move your data outside of that datacenter unless they're dealing with a business continuity situation. This should reduce the latency you'll experience over time, but will also increase costs since there is usually an up-charge for this service.

However, network latency is not the only network-related performance issue. In many instances, network latency can occur within the enterprises itself. Older networking equipment, as well as lack of firmware and software updates, can mean that the latency is self-inflicted.

The best path here is to install and test networking performance management tools, which should show you where the issues are, inside and outside of the firewall. In many instances, the use of cloud just puts a spotlight on existing networking problems, and does not cause the problems themselves.

POOR DESIGN

As we discussed above, many of those tasked with migrating systems from the local datacenter to public clouds consider this an A-to-A port with few modifications required. While the system may run fine, if it's not localized to leverage the performance features of the host cloud, you're not getting the best performance bang for your cloud buck.

Moreover, public cloud-hosted performance should be considered in the design of your system, including how to deal with resource provisioning and deprovisioning, platform optimization, etc. Most public cloud providers get paid regardless of whether or not your system is optimized. However, you'll pay larger bills and suffer poor performance. Those issues will diminish with a bit of good system design that leverages the native features of the public cloud.

Some emerging best practices include:

- Understand the best approaches to leverage platforms on your public cloud provider. In many instances, they have special tricks that you can add to your code that will provide noticeable savings in performance, stability, and cloud usage costs.
- Performance is not just about use of the cloud-native APIs, but also the design of the application as a whole. In many instances, legacy applications are poorly designed. While they got by on-premises operating on single tenant platforms, in the cloud they are outed for being poorly designed. You need to consider the best approach to cloud application design. As you remediate the existing on-premises application designs into cloud-native versions, you need to consider a better holistic design as well.

UNDERSTAND THE APPROACH TO TENANT MANAGEMENT

Finally, you should understand the tenant management features of the public cloud provider, and work around any quirks. All clouds do not approach tenant management in the same way. It's helpful to understand how the cloud provider deals with multitenancy at an architecture level so you can optimize your use of the public cloud resource.

For instance, consider the approach to multiplexing I/O, such as reading and writing to storage. Or, the approach to dealing with queueing, database reads and writes, and prioritization for the CPU, memory, and storage.

Guess what? The public cloud provider is unlikely to provide you with this information. Instead you have to depend upon other users to help you understand what the tenant management approach is, and how to change applications to take full advantage.

The idea is to alter your approach to when, how, and the size of the chunks of resources you provision to support your system processing. For example, you may find that provisioning smaller amounts of resources at a time does not task the tenant management system as much as allocating one large chunk. The idea is to understand the quirks of the cloud, and work around them to optimize performance.

PERFORMANCE PLANNING

As we become more experienced with public clouds, the ability to manage performance should improve. Cloud providers should also learn and evolve their cloud computing offerings over time to provide better performance and scalability. However, for the next few years, cloud computing performance management will be a necessary skill that most enterprises will need.

The bottom line is that if you're not willing to plan for performance, then you'll probably have poorly performing public cloud workloads. More likely, you'll have huge cloud bills that get bigger every month.

Creating a performance plan requires a few core steps, including:

- Defining the performance expectations of the business, include response time, uptime, recovery time, and costs that the users are willing to pay.
- Defining the multitenant approach that your pubic cloud provider leverages, which includes approaches to I/O management, etc. Create a sub-plan that focuses on how to leverage the cloud multitenant approach so the workloads are optimized.
- Defining how the applications and data stores should be modified to take advantage of the public cloud you picked. This will typically look like T-shirt sizes—small, medium, large, and extra-large—in terms of the level of effort and cost.
- Defining a performance management plan, including tools, talent, and processes for checking that cloud performance is meeting expectations.

While this seems complex, it's really something we go through each time we change platform technology, which we've done many time in the past. This time, however, there are many more moving parts. These parts need to align properly, or performance will be an ongoing issue.

REFERENCES

1. "Survey: Companies are Bullish on Cloud Analytics, But Need to Speed Up the Pace," Teradata, 2018; https://www.teradata.com/Press-Releases/2018/Survey-Companies-are-Bullish-on-Cloud-Analyt.

ABOUT THE AUTHOR

David S. Linthicum is the Chief Cloud Strategy Officer at Deloitte Consulting, and was just named the #1 cloud influencer via a recent major report by Apollo Research. He is a cloud computing thought leader, executive, consultant, author, and speaker. Linthicum has been a CTO five times for both public and private companies, and a CEO two times in the last 25 years. Contact him at david@davidlinthicum.com.

TOSCA Solves Big Problems in the Cloud and Beyond

Paul Lipton
CA Technologies

Derek Palma
Vnomic

Matt Rutkowski
IBM

Damian A. Tamburri
TU/e - JADS

TOSCA, the "Topology and Orchestration Specification for Cloud Applications" offers an OASIS-recognized, open standard domain-specific language (DSL) that enables portability and automated management of applications, services, and resources regardless of underlying cloud platform, software defined environment, or infrastructure. With a growing, interoperable ecosystem of open source projects, solutions from leading cloud platform and service providers, and research, TOSCA empowers the definition and modeling of applications and their services (microservices or traditional services) across their entire lifecycle by describing their components, relationships, dependencies, requirements, and capabilities for orchestrating software in the context of associated operational policies. The authors introduce important TOSCA concepts and benefits in the context of commonly understood cloud use cases as a foundation to future discussions regarding advanced TOSCA concepts and additional breakthrough issues.

Tosca is a brilliant opera by Italian composer G. Puccini. Much like this operatic masterpiece, OASIS TOSCA, the "Topology and Orchestration Specification for Cloud Applications," also offers the possibility of brilliant harmony and orchestration, in this case across hybrid, technologically heterogeneous IT environments.

TOSCA is an OASIS-recognized, open standard domain-specific language (DSL)[1] that enables application and service portability, as well as automated operational management of applications, services and resources (e.g. networks, storage, containers, clusters, microservices, etc.) regardless of underlying cloud platform, software defined environment, or infrastructure. TOSCA accomplishes this by enabling machine and human-readable models that can drive the orchestration of these applications, services, and resources across their entire lifecycle, from initial deployment

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

through various lifecycle stages such as scaling/un-scaling and finally to un-deployment. TOSCA enables the interoperable description of application and service components, relationships, dependencies, requirements, and capabilities in the context of their associated operational policies. From a business viewpoint, TOSCA expands customer choice, reduces cost, and increases business agility across the application lifecycle by leveraging powerful and in some respects unique modeling concepts. The synergy between all of these benefits accelerates overall time-to-value.

TOSCA is an Open Standard developed within OASIS (www.oasis-open.org), which is an internationally recognized organization for the development of open standards consisting of approximately 600 companies, government agencies, and academic organizations from around the world represented by over 5000 participants spanning over 65 countries. This well-governed organization empowers the TOSCA Technical Committee (TC) to operate transparently with the help of nearly 200 technical experts representing a growing, interoperable eco-system of open source projects such as OpenStack, software and service vendors, solution providers, and research organizations (see Figure 1 for a representative subset). The TOSCA TC also cooperates with other key international standards and research initiatives, e.g., ETSI NFV, EU FP7, and EU H2020.

Figure 1. TOSCA Technical Committee organizations (a representative subset).

The TOSCA TC ensures that the use cases that help drive the TOSCA standard are derived from the experiences of real-world TOSCA implementers and researchers, who are expanding the scope and application of TOSCA to new domains, e.g., container management and clustering, the Internet-of-Things (IoT), multi-cloud orchestration,[1] hybrid cloud applications with microservices distributed across heterogeneous cloud platforms, BigData,[2] Analytics, Cognitive Computing, and more. Many of these use cases and benefits are described in the latest OASIS TOSCA Final Deliverable, which at the time this was written was OASIS TOSCA Simple Profile in YAML version 1.0,[3] although work on OASIS TOSCA Simple Profile in YAML version 1.1,[4] which contains a notable enhancement discussed further below, is already nearing completion.

Because TOSCA is designed to express the general semantics required to orchestrate any set of components or resources across specific lifecycle states with complex interdependency constraints, it can also be used to solve a diverse set of IT orchestration problems. The original focus of TOSCA was application-centric, but the TOSCA TC is actively expanding into domains such as Software Defined Networks (SDN) and Network Function Virtualization (NFV). In cooperation with the ETSI NFV Industry Work Group and other NFV communities including open source, the TOSCA TC is already well along in the development of a TOSCA Simple Profile for NFV, which extends TOSCA to the world of software-defined network services, such as virtual

network routers, switches, and multi-function devices. We intend to cover this work in a subsequent article. Finally, we intend to introduce considerations regarding the use of TOSCA in more diverse domains, as well as discussions of future developments and advanced TOSCA concepts, in future discussions that will build upon the abstractions and conceptual framework introduced in this article.

WHY USE TOSCA?

Cloud platforms, both IaaS and PaaS, have already heavily invested in the development of a wide range of platform-specific, optimized, and mature APIs and protocols designed for deployment and packaging of applications on their proprietary or open platform. These are unlikely to be abandoned or deprecated.

TOSCA accepts this reality, and offers a more useful, higher level of abstraction by enabling application and service portability in the form of a recognized Open Standard that is a multi-platform, cross-technology, human and machine-readable, YAML-based topological modeling language. TOSCA frees the application/service developer from the need, complexity, and cost of having to learn and write different, complex code to control various platform-specific APIs and protocols to package and/or deploy an application/service, an expensive and error-prone process, for each and every application/service to be deployed.[5]

Rather, in TOSCA, the burden of leveraging platform-specific APIs and protocols falls upon the TOSCA cloud orchestrator software; built and optimized to work with the cloud platform and with that platform's well-established, platform-optimized, continuously-evolving APIs and protocols, not on the application or service owner/developer. With TOSCA, the solution/application owner is not required to maintain or leverage any API or protocol expertise, open or proprietary. TOSCA orchestrators enable Cloud Service providers to meet customer demand for portability at the highest level of abstraction using an extensible model while continuing to differentiate themselves and their platform's capabilities.

TOSCA models of an application and any constituent services is sufficient regardless of platform destination. It is important to note that TOSCA is not a "lowest common denominator" approach. Quite the opposite. It enables TOSCA orchestrators to automatically leverage the unique capabilities that EVERY TOSCA compliant cloud has to offer. TOSCA orchestrators can optimally match application and service component requirements in the TOSCA model to the capabilities and desired characteristics of other components, services, and host environments (including containers and virtual machines) provided by the platform technology and cloud platform provider. For example, a TOSCA application's components may express requirements for certain versions, configurations, or desired characteristics from database, containers, subordinate services, etc.

TOSCA is also application architecture independent. It specifically addresses the intrinsic and quite necessary dependencies and interrelationships implicit in successful distributed applications and services regardless of architecture style, e.g., microservices, serverless, etc. It does so while addressing the challenges of a diverse and ever-changing world of containers technologies, e.g., hypervisor-based or "leaner" containers such as Docker, as well as container automation and tools, e.g., Kubernetes, Mesos, or Spawn. In all these cases, TOSCA provides unification and stability to the rapidly shifting technology "sands beneath our feet" by handling this diversity in a consistent fashion while offering the enduring value of a technology-neutral, international standard.

Using a non-trivial Docker-based application or microservice as an example, such an application or microservice needs to be understood and modeled in context of other services (micro and/or otherwise), traditional and/or containerized in some fashion, across heterogeneous cloud platforms (IaaS and/or other service models (including PaaS)). TOSCA already addresses this need, and in the newest version of TOSCA, v1.2, the TOSCA Technical Committee is currently working to extend its capabilities in this area by focusing with greater descriptive depth on opaque artifacts such as Docker containers in a fashion that is standardized and technology independent. Docker containers also need to be modeled with important operational policies, e.g., scaling and placement (affinity) that TOSCA already defines. This can empower and be appropriately acted

upon with the coordination of orchestrators with container orchestrators such as Mesos and Kubernetes.

Lastly, it is important to note that the TOSCA standard is designed to address not merely deployment, but the entire application lifecycle while supporting Operational Policies declaratively, e.g., scaling, placement, etc. It is also notable that the TOSCA Simple Profile in YAML Version 1.1 abstracts entity lifecycle operations allowing TOSCA orchestrators to fulfill component creation, installation and configuration operations using existing investments in scripting/imperative technologies, such as Puppet and Chef, within a TOSCA environment, as needed, further reducing cost and speeding adoption of TOSCA.

TOSCA MODELING CONCEPTS

TOSCA uses a simple and easy to understand DSL.[1,6] TOSCA is designed to express the kinds of entities, relationships, constraints and semantics encountered across the diverse IT domains. TOSCA Service Templates are reusable and composable models that can increase return on investments by making it easy to deploy, configure, and operationally manage more valuable and complex applications throughout their lifecycle from TOSCA models of existing applications and services.[3–4]

TOSCA Service Templates describe the overall application/service topology, and consist of a graph of Node Templates that model the application/service components and Relationship Templates that model the relationships and cardinalities between those components, e.g., various kinds of cloud consumer-provider dependencies such as containment and connection. Policy Templates specify the necessary operational constraints on specific parts of the application/service topology.

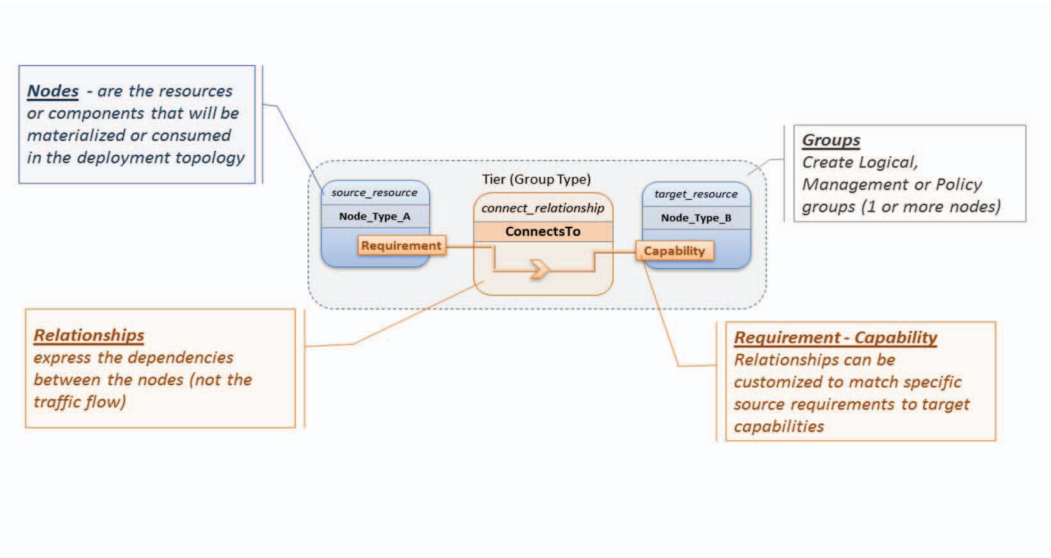

Figure 2. Key TOSCA Modeling Concepts Illustrated with the "*ConnectsTo*" Relationship type.

TOSCA provides a type system to concisely express the schema of the template entities, e.g., node types, relationship types, requirement types, capability types, artifact types and policy types (see Figure 2). TOSCA orchestrators and tools process and validate Service Templates by referring to the respective type for each template entity. Naturally, essential types expressing fundamental concepts such as storage, network, and compute are defined and extended to higher-level abstractions, e.g., BlockStorage, Webserver, application and infrastructure components, and much more.

Extensible, normative types have been defined and driven by use cases that reflect the considerable experience of the many TOSCA implementers. We encourage the reader to read the publicly

available specification[3] to gain a full appreciation, but just one more example here to make it more real: TOSCA Artifact Types can represent packages and files, such as VM deployment images or containers, used by a TOSCA orchestrator or for deployment and other purposes. Currently, artifacts are logically divided into three categories:

- Deployment Types: artifacts used during deployment, e.g., TAR files, RPMs, etc.
- Implementation Types represent imperative model logic, e.g., scripts, chef, puppet, etc., and are used to implement TOSCA Interface operations.
- Runtime Types: artifacts used during runtime by an application service or component, e.g., containers or VM images.

Both node and relationship types may define lifecycle operations to direct the behavior of the orchestration engine when instantiating a service template. For example, a node type might provide a 'create' operation to handle the creation of that component instance at runtime, or a 'start' or 'stop' operation, as needed. These lifecycle operations can be backed by implementation artifacts such as scripts, Chef Recipes, or APIs and management frameworks that implement the actual behavior.

The orchestration engine uses the Relationship Templates between template nodes to derive the order of component instantiation or operations during lifecycle events. For example, during the instantiation of a two-tier application that includes a web application that depends on a database, an orchestration engine would first provision a container and then invoke the 'install,' 'configure,' and start operations on the database component to install and configure the database on the container, and it would then invoke the respective lifecycle operations of the web application to install and configure the application (which includes configuration of the database connection). Different packaging and container types are supported by providing specific lifecycle operations or prebuilding configuration state into the deployment artifacts.

It is important to note that TOSCA elevates the concepts of requirements and capabilities to first-class objects. In practical terms, this enables TOSCA orchestrators and tools to dynamically match component requirements to capabilities provided by other components, services, or the underlying platform itself. For example, a Node Type may have the capability (simply expressed as tosca.capabilties.Container) of acting as a container (or a host for) one or more other declared Node Types. Similarly, a requirement constraining the DB to be NoSQL could be declaratively expressed. Operational policies such as placement affinity or anti-affinity, or resource constraints such as memory or bandwidth, could also be expressed. In fact, TOSCA enables models and dependencies to be validated, even in advance of deployment, in order to ensure application-aware, policy-aligned configuration, deployment and operational semantics for applications and their services with greater speed and at lower cost.

Lastly, TOSCA also contains a number of other important features designed to simplify and extend the utility of TOSCA Service Templates:

- Inputs can be defined in the Service Template to allow the user to provide specific parameter values at deployment time
- Nodes can left unresolved in the Service Template to:
- Allow the TOSCA orchestrator to resolve the node with a concrete instance or entire sub-topology that meets the requirements of the Service Template
- Allow users to select or provide alternative or equivalent node implementations in different environments
- The Service Template has a packaging structure that allows it to be published as a single unit, but to refer to deployment artifacts in different repositories

A NON-TRIVIAL TOSCA WALKTHROUGH

A TOSCA orchestrator processes a Service Template as a declarative specification of the desired state of the application or service topology.[4] It allows users to delegate as much or as little decision-making to the TOSCA orchestrator as needed, and enables the orchestrator to make intelligent decisions that provide the application an opportunity to maximally leverage the unique differentiators of the cloud platform provider. It also provides value to cloud platform providers, enabling them to more easily differentiate and innovate for their users, rather than being forced to settle for technical and business limitations compelled by the use of a generic, lowest-common denominator generic API or proxy approach.[3–4]

Multiple vendors demonstrated these inherent TOSCA advantages and many other important TOSCA concepts using a non-trivial, real-world example at the OSCON conference in July 2015, where they showed how three tiers could be assembled and composed into one complete application from different TOSCA Service Templates inherited from different catalogs distributed on completely different infrastructures and heterogeneous cloud platforms.

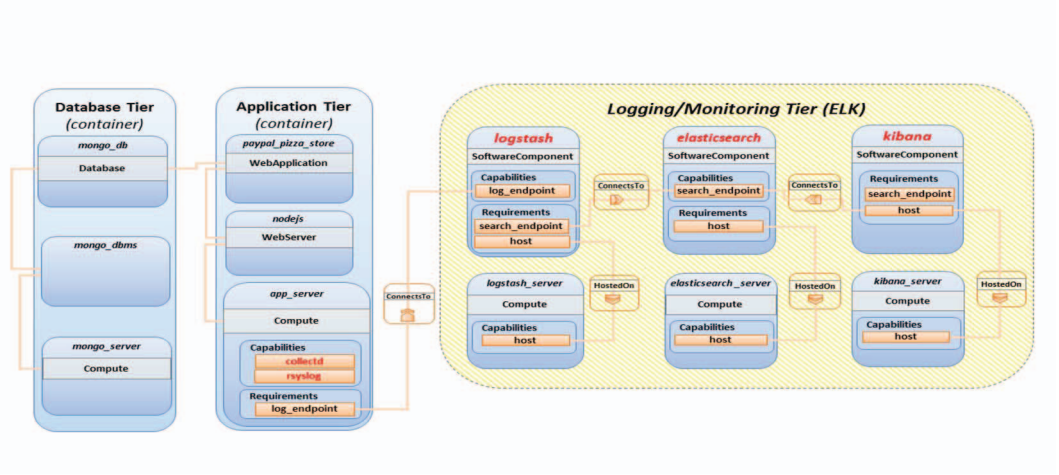

Figure 3. Service Templates: An example.

This demonstration established a Logging and Monitoring Service composed of the popular *ElasticSearch*, *LogStash* and *Kibana* (ELK) open source components. This type of service is often connected to a multi-tier application for which further monitoring and operations management is needed (see Figure 3). In this example, the *app_server* compute node has additional requested capabilities for *collectd* and *rsyslog* in addition to the log_endpoint requirement that allows it to connect to service template via the ConnectsTo normative TOSCA relationship.

Each tier consists of a graph of node templates interconnected with different relationship templates. The two most common relationships templates are HostedOn and ConnectsTo. All nodes must either be hosted on some other node or be handled by the cloud platform in order to be instantiated. The requiring node (hosted component) refers to the host capability of the hosting node. This capability may have properties that can be specified by the requiring node, such as the memory required.

The HostedOn relationship results in hierarchies of components as seen in Figure 3. For example, the *mongo_db* component is hosted on the *mongo_dbms* (the mongo DB instance), and the *mongo_dbms* component is hosted on *mongo_server*, which is a compute node that could be a physical server, virtual machine or some kind of container. A TOSCA orchestrator would traverse the topological graph in order to create the nodes in the appropriate order; *mongo_server* first, *mongo_dbms* next, and *mongo_db* last.

A TOSCA orchestrator typically supports "root" nodes like the compute node by automatically provisioning it, and then transitioning it to a usable state using a configuration like the one specified by the *mongo_server* node template. This avoids the need for the user to define all such details in the Service Template. For the other application nodes, a TOSCA orchestrator would execute their standard lifecycle operations, such as create, configure, and start.

The ConnectsTo relationship is used to specify that a node requires a connection to another node in the topology. The requiring node refers to the capability in the node it requires, which in the case of ConnectsTo is an end point (also modeled as a capability). As with any capability in TOSCA, the requiring node can set parameters defined in the target capability. A TOSCA orchestrator will resolve ConnectsTo relationships by ensuring that the target node is instantiated and in the appropriate state before it completes the fulfillment of the relationship.

The definition of the ConnectsTo relationship requires that source and target components of the relationship be initiated and in specific operating states. As mentioned above, the orchestrator would fulfill the appropriate HostedOn relationships for the source and target. Relationships can define lifecycle operations in a similar fashion to nodes, but these can be interwoven with the target node lifecycle operations. For example, the preconfigure operation executes before the configure operation on the target, allowing a special step to be performed or specific information to be propagated to the target node instance before it is completely configured.

Not shown here, but noted above, the TOSCA Simple Profile in YAML Version 1.1 enables TOSCA orchestrators to fulfill component creation, installation and configuration operations using existing investments in workflow (a.k.a. scripting or imperative model technologies), such as Puppet and Chef, within a TOSCA environment, as needed, further reducing cost and speeding adoption of TOSCA. However, when using workflow within a TOSCA Service Template, as opposed to TOSCA's normative, declarative orchestration, the inevitable tradeoffs must be considered.

For example, use of imperative model workflow within a Service Template requires that the user takes responsibility for all potential use of the manually created workflow, and to do so for all topologies, environments, and cloud platform variations. This is likely to impose higher maintenance and complexity cost over time. Thus, TOSCA's normative, declarative orchestration is preferred, when possible, because it maximizes the Service Template portability and reusability across a wide range of the most common automation use cases.

A TOSCA Service Template, or *blueprint,* is a YAML encoding,[1] which contains the specification of a topology template comprised of inputs, outputs and node templates which make up the service topology. The service template fragment in Figure 4 represents the deployment of open source monitoring components discussed more abstractly above, in this case *Elasticsearch*, *Logstash* and *Kibana* (ELK).

Figure 4 shows, in YAML, that the type specified in the *Logstash* node template is the node type which defines the schema of the instantiated Logstash node template. The *Logstash* template contains a host requirement which points the *Logstash* server, which it will be deployed on. It also has a search endpoint requirement to connect to the *Elasticsearch* server, defined in a separate node template (not shown). The search endpoint relationship specifies the lifecycle operation, called *pre_configure_source*, which executes in order to configure the *Elasticsearch* connectivity from the *Logstash* node. The Interfaces keyword specifies the lifecycle and respective lifecycle operations. Each lifecycle operation, e.g., configure or start, is specified with the artifact used to execute it. Additionally, input values may be passed as explicit parameters to each operation.

The *Logstash* server node template indicates that it is a tosca.nodes.Compute type, which is a TOSCA normative node type that defines the named *host* capability (which is of type tosca.capabilities.Compute.Container) required by the *Logstash* node template mentioned above. It also has a capability named *os*, which allows it to fulfill specific OS type, version and distro requirements.

```
tosca_definitions_version: tosca_simple_yaml_1_0_0

description: >
  This TOSCA simple profile deploys nodejs, mongodb, elasticsearch, logstash and kibana each on a separate server...

imports:
  - paypalpizzastore_nodejs_app.yaml
  - elasticsearch.yaml
  - ...

topology_template:
  inputs:
    my_cpus:
      type: integer
      description: Number of CPUs for the server.
      constraints:
        - valid_values: [ 1, 2, 4, 8 ]
    ...

  node_templates:
    logstash:
      type: tosca.nodes.SoftwareComponent.Logstash
      requirements:
        - host:
            node: logstash_server
        - search_endpoint:
            node: elasticsearch
            capability: search_endpoint
            relationship:
              type: tosca.relationships.ConnectsTo
              interfaces:
                Configure:
                  pre_configure_source:
                    implementation: Python/logstash/configure_elasticsearch.py
                    inputs:
                      elasticsearch_ip: { get_attribute: [elasticsearch_server, private_address] }
      interfaces:
        Standard:
          create: Scripts/logstash/create.sh
          start: Scripts/logstash/start.sh
     logstash_server:
      type: tosca.nodes.Compute
      capabilities:
        host:
          properties: *host_capabilities
        os:
          properties: *os_capabilities
```

Figure 4. TOSCA blueprint: A basic example.

SIDEBAR: TOSCA TOOLS AND RESOURCES

The TOSCA standard has been taking off in many areas, spanning from seamless multi-cloud research and practice, to open-source, to medical computing, to information interchange. Many tools are emerging in these domains that accompany TOSCA and allow practitioners and researchers to harness and evaluate the benefits brought about by TOSCA in action. Naturally, the TOSCA Technical Committee does not endorse any particular implementation, nor do the authors claim to Table 1 below is a complete or authoritative list of implementations. It is more in the nature of a representative subset. In fact, we have often been "surprised" to learn about new implementations appearing, quite organically, in the open source community and privately within major corporations.

Table 1. Overview of TOSCA open source projects and tools.

Tool/Project Name	Brief Description	Online Reference
Alien4Cloud	Type and workflow designer featuring quasi-full TOSCA support and works with ARIA and Cloudify as well. Standard workflow description 1.1 is going to be released soon.	http://alien4cloud.github.io
ARIA	Multi-cloud orchestration featuring Microsoft Azure cloud, Amazon Web-Services, VMware and open-stack. Spin-off open-sourced from Cloudify. ARIA consumes TOSCA models to deploy onto multiple clouds offering seamless multi-cloud management.	http://ariatosca.org
Cloudify	Service Orchestration & Management platform supporting TOSCA with ad-hoc conformance but in-depth general support to all essential TOSCA constructs.	http://getcloudify.org
DICER	Model-Driven continuous architecting tool prototype specific for big data applications deployment (currently featuring Apache projects Spark, Storm, Hadoop and Oryx2) [2,6] – key constituent part of the DICE H2020 project (http://dice-h2020.eu/).	https://github.com/dice-project/DICER
heat-translator	Heat-Translator is an OpenStack project that provides an extensible command line tool and library that accepts a TOSCA Service Template in-memory "graph" generated by the tosca-parser project library as input to produce a Heat Orchestration Template (HOT), which can be deployed by OpenStack. It is easily extended to produce other output formats, e.g., a conversion from TOSCA to Kubernetes YAML.	https://pypi.python.org/pypi/heat-translator
Indigo	DataCloud is an open source data and computing platform targeted at scientific communities, deployable on multiple hardware and provisioned over hybrid, private or public, e-infrastructures. Indigo is part of the CERN project.	https://www.indigo-datacloud.eu/
Murano and Community App Catalogs	OpenStack application cataloguing tools, featuring support to all openStack enabled clouds and harnessing TOSCA for description, integration and portability dynamics.	https://apps.openstack.org/
	OPEN-O is an open source project that enables telecommunications and cable operators to effectively deliver end-to- end services across Network Functions Virtualization (NFV) Infrastructure, as well as Software Defined Network (SDN) and legacy network services.	
OPNFV Parser	This platform provides a framework for integration of a wide range TOSCA tooling, e.g., orchestraters, modelers, converters, and other tools, for the NFV community.	https://wiki.opnfv.org/
OpenTOSCA	Self-service portal featuring TOSCA modeling tool and runtime environment based on the original XML-based standard.[5,7]	http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php
Senlin	OpenStack clustering and policy management project capable of dealing with all related TOSCA constructs and operational semantics (e.g., Event-Condition-Action policy design pattern).	https://wiki.openstack.org/wiki/Senlin
Tacker	OpenStack open-source project specific to NFV design, management and operation. Tacker takes TOSCA Models exclusively.	http://wiki.openstack.org
tosca-parser	This *standalone* TOSCA Parser Pypi library is also available from the OpenStack community. It produces an "in-memory" graph that can be used by other tooling such as translators, orchestrators and modelers.	https://pypi.python.org/pypi/tosca-parser
tosca-translator	This *standalone* Pypi library is a TOSCA language translator that can take the graph produced by the standalone parser and produce another DSL. Original target was Heat, but proofs-of-concepts have been written for other targets such as Kubernetes.	https://pypi.python.org/pypi/heat-translator
Ubicity Validator	TOSCA Service Template model validator	http://ubicity.com/types.html

CONCLUSION AND FUTURE WORK

TOSCA is an approved, OASIS standard topology and orchestration language with a large and growing eco-system of open source and proprietary implementations, as well as active research effort. The TOSCA Technical Committee (TC) has attempted to provide an admittedly incomplete set of links and references to this growing universe of "TOSCA-teers" on its public page,[7] and the Technical Committee is already hard at work on further improvements to TOSCA itself, starting with further improvements to interoperability using the OASIS Test Assertion Language standard as a foundation. While TOSCA already intrinsically supports fundamental container and microservice concepts, there is keen interest within the TOSCA TC to make further improvements in order to maximize the value of TOSCA using such technologies and architectures in the future. Also, there is significant work in enabling runtime generation of TOSCA blueprints to further support tooling across the entire application lifecycle and adaptable, software-driven environments that are even more flexible and resilient, which also complements the work being done to enhance the monitoring support implicit in the language.

The potential for TOSCA, beyond portability and automated management of applications, services, and resources regardless of underlying cloud platform, software defined environment, or infrastructure, may lie in its flexibility. Over time, the flexibility of TOSCA may enable it to provide its unique benefits to other domains outside of cloud, including Internet of Things, large-scale distributed applications focused on emerging technologies such as BigData, Hyper-Ledger,

and so on. The TOSCA TC already has several teams working closely with other leading standards organizations such as ETSI, e.g., to produce a TOSCA Simple Profile for NFV (Network Function Virtualization) this domain will be the subject of a more detailed future article.

From a research perspective, we plan to explore the runtime aspects of TOSCA and how it can be augmented with annotations and notations that reflect its runtime modelling form such that more advanced cloud management can ensue. Moreover, we plan to explore deeper the "intent" modelling aspects hardcoded within TOSCA such to understand the theoretical aspects of such novel modeling approach, its tenets and challenges. Finally, we plan to elaborate more on the support that TOSCA brings about in terms of continuous architecting, i.e., the activity of refining software architectural and infrastructural blueprints continuously and iteratively.[8]

REFERENCES

1. E.D. Nitto et al., "Supporting the Development and Operation of Multi-cloud Applications: The MODAClouds Approach," *Proc. of International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC 13), 2013, pp. 417–423.
2. M. Artač et al., "Model-Driven Continuous Deployment for Quality DevOps," *Proceedings of the 2nd International Workshop on Quality-Aware DevOps* (QUDOS 16), 2016, pp. 40–41.
3. *OASIS TOSCA Simple Profile in YAML version 1.0*; http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/cs01/TOSCA-Simple-Profile-YAML-v1.0-cs01.pdf.
4. *OASIS TOSCA Simple Profile in YAML version 1.1*; http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/.
5. T. Binz et al., "Portable Cloud Services Using TOSCA," *IEEE Internet Computing*, vol. 16, no. 3, 2012, pp. 80–85.
6. M. Guerriero et al., "Towards a model-driven design tool for big data architectures," *Intl. Workshop on Big Data Software Engineering* (BIDSE@ICSE 16), 2016, pp. 37–43.
7. U. Breitenbücher et al., "Vinothek - A Self-Service Portal for TOSCA," *6th Central European Workshop on Services and their Composition* (ZEUS 14), 2014.
8. L.J. Bass, I.M. Weber, and L. Zhu, *DevOps — A Software Architect's Perspective*, Addison-Wesley, 2015.
9. S. Dustdar et al., "Programming Directives for Elastic Computing," *IEEE Internet Computing*, vol. 16, no. 6, 2012, pp. 72–77.
10. *OASIS TOSCA Technical Committee public page*; https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.
11. P. Clements et al., *Documenting Software Architectures: Views and Beyond*, Addison-Wesley Professional, 2002.

ABOUT THE AUTHORS

Paul Lipton is VP of Industry Standards and Open Source at CA Technologies where he coordinates strategy and participation in these areas. He co-chairs the TOSCA Technical Committee, and also serves on the Board of Directors of OASIS, the Eclipse Foundation, the OMG, and the DMTF. He is an approved US delegate to the SO/IEC JTC 1 initiatives focused on cloud standards, Internet of Things, and Smart Cities. Contact him at Paul.Lipton@ca.com.

Derek Palma is the founder and CTO at Vnomic, a pioneer in the declarative delivery and governance of complex applications on software defined infrastructures. He is an expert in model based systems and desired-state automation, co-leader of the TOSCA Instance Model group, and OASIS TOSCA co-editor. Contact him at dpalma@vnomic.com.

Matt Rutkowski is a senior engineer and master inventor at IBM and has worked to develop open infrastructure and industry standards and open source for over 15 years in areas such as Government, Banking and Digital Media and Entertainment. Most recently, he has been working on cloud, serverless technology, and software security standards. He was lead editor for the OASIS IDCloud TC, founder and Co-Chair of the DMTF Cloud Auditing Work Group and also is Co-Chair of the OASIS TOSCA Interop Subcommittee and a Technical Committee contributor/editor. Contact him at mrutkows@us.ibm.com.

Damian A. Tamburri is an assistant professor at the Technical University of Eindhoven and the Jheronimus Academy of Data Science in s'Hertogenbosch, The Netherlands. He serves as active voting member of the TOSCA Technical Committee. His current research interests lie mainly in advanced software architecture styles (e.g., SOA, Big-Data, etc.), advanced software architecting methods (e.g., MDA, continuous architecting and DevOps), and social software engineering (Socio-technical congruence, Measuring Social Debt, etc.) and their investigation by means of Empirical Software Engineering. Contact him at d.a.tamburri@tue.nl.

Privacy-preserving Image Processing in the Cloud

Zhan Qin
University of Texas at San Antonio

Jian Weng
Jinan University

Yong Cui
Tsinghua University

Kui Ren
State University of New York at Buffalo

Millions of private images are generated in various digital devices every day. The consequent massive computational workload makes people turn to cloud computing platforms for their economical computation resources. Meanwhile, the privacy concerns over the sensitive information contained in outsourced image data arise in public. In fact, once uploaded to the cloud, the security and privacy of the image content can only presume upon the reliability of the cloud service providers. Lack of assuring security and privacy guarantees becomes the main barrier to further deployment of cloud-based image processing systems. This paper studies the design targets and technical challenges lie in constructing cloud-based privacy-preserving image processing system. We explore various image processing tasks, including image feature detection, digital watermarking, content-based image search. The state-of-the-art techniques, including secure multiparty computation, and homomorphic encryption are investigated. A detailed taxonomy of the problem statement and the corresponding solutions is provided.

Motivated by the rapid growth of image processing and data mining techniques, more and more image processing based applications are deployed in various end-users' devices. For example, content-based image search, digital watermark verification, and so on. The consequent massive image processing tasks bring enormous computation overhead to data owners. To solve this problem, more and more users are outsourcing the "expensive" tasks to cloud computing platforms. In one such cloud computing platform, Cloud Service Provider (CSP) offers a pay-per-use business model, which lets individual users use robust computation power in the cloud while saving time and costs on setting up corresponding infrastructures.[1] In fact, not only individual or small business data owners but Internet giants like Microsoft and Yahoo are also attracted by the benefits brought by cloud computing and authorize some services to third-party cloud computing platforms. For example, several types of data searching tasks in Microsoft Bing have been outsourced to Wolfram.[2]

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD

However, the participation of a third-party cloud computing platform also increases the vulnerability of private data, e.g., potential data breaches and losses. Under current cloud architecture, the content of outsourced image data will inevitably be leaked to CSPs. In this case, the leaked content might be sensitive information such as the data owner's personal identity, home address, or even financial records. Moreover, even if we assume CSPs are completely honest and could be trusted to have data owners' private information, such privacy leakages still happen. In fact, the cloud server is usually considered as a low-qualified locker rather than a strong bank deposit box.[3] The cloud computing platform suffers from more security threats compared with a traditional network server. For instance, a severe vulnerability in cloud servers is the sharing of computing resources: flaws in System Virtual Machine (SVM) software have frequently been discovered and exploited to attack cloud servers in recent years.[4] Nevertheless, private data leakage in the public cloud happens very often due to the improper configuration and maintenance by CSPs. In a nutshell, privacy concerns over outsourced data have become the main barrier to the further development of cloud computing platforms.

In recent years, secure image data processing has grown rapidly as a research field and attracted attention from both academia and industry. In practice, many fancy image-processing applications require computational power beyond the limit of mobile devices. For example, 3D structure reconstruction needs massive computational power for image feature detection and matching. In this area, the main research direction lies in the detection of image features over ciphertext domain.[5] Many encryption techniques are applied or adjusted to protect image data privacy while enabling visual feature extractions. Qin and colleagues proposed a global image feature detection mechanism for color histogram-based descriptors detection.[6] The authors utilized a Somewhat Homomorphic Encryption (SHE) scheme to enable the computation of diverse color descriptors in the MPEG-7 standard over the ciphertext domain. These features are further utilized as basic building blocks for services such as image matching and semantic tag generation. Hsu and colleagues proposed a local feature detection mechanism for Scaler Invariant Feature Transform (SIFT),[7] which utilizes the Paillier encryption scheme to enable the computation of SIFT features over ciphertext domain. In another work,[8] the authors analyzed different scaling ratios by adjusting fixed point numbers in the proposed scheme. However, all these works suffer from the high computational complexity brought on by homomorphic operations, especially for those who perform relatively complicated algorithms like SIFT. Qin and colleagues solve this problem by utilizing a multi-server structure to enable SIFT algorithm over encrypted data.[9] Another thriving research direction is secure digital watermarking, which enables outsourcing the time-consuming tasks of generating digital watermark without compromising the privacy of the image content. Two types of approaches have been proposed: asymmetric watermarking[10] and zero-knowledge watermark detection[11]. However, most existing works still suffer from the high computational complexity on both user and cloud side.

Lu and colleagues proposed an orthogonal research direction, the secure image retrieval mechanism is proposed,[12] which enables applications such as location-based detection. It offers flexible approaches to manage private image datasets online, and the features extracted from images are encrypted in a distance-preserving scheme to enable direct comparisons for similarity evaluation.[12] In the work of Erkin and colleagues, the current image search indices are encrypted while achieving searching functionalities with efficiency.[13] However, in a practical privacy-preserving computation scenario, all the existing works are very difficult to achieve the security requirements and practical efficiency performances at the same time.[14]

This article introduces and formulates diverse image processing tasks in a general image computation outsourcing model, including image feature detection, digital watermarking, and content-based image search. We discuss state-of-the-art techniques, including secure multiparty computation and homomorphic encryption. Finally, we provide a detailed taxonomy of the problem statement and corresponding solutions.

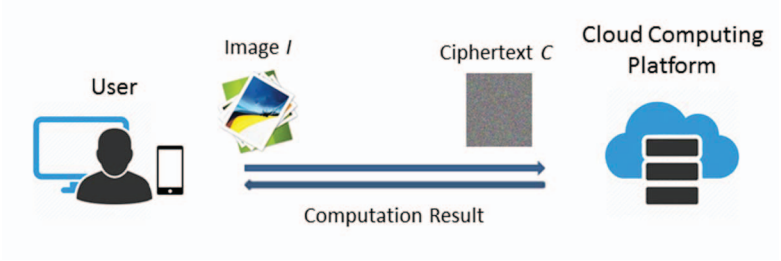

Figure 1. System model.

PRIVACY PROTECTION IN IMAGE DATA PROCESSING

System Model and Workflow

System Model

As shown in Figure 1, the proposed system consists of two main entities: the Cloud Computing Platform (CCP) and the user. The user is a data owner who holds massive image data and intends to outsource the image processing tasks to the CCP. In this setting, a user utilizes the CCP as a complementary resource for his limited computational power and also outsources complicated image processing tasks to the CCP. Meanwhile, users need to protect the privacy of their data. For example, hospitals are under an obligation to protect patients' records such as medical images and profiles. In this case, to protect a user's privacy, he or she has to encrypt the image data before outsourcing to the CCP. Meanwhile, the entity CCP is composed of a set of cloud servers assumed to be honest but curious. It can only access the encrypted image data uploaded by users and perform the corresponding image processing algorithms over the ciphertext domain. After that, the CCP returns the requested results in the form of ciphertext back to a user. Finally, a user can use her private key to decrypt the returned results. Throughout the process, the CCP should not have any access to the content or results of the user outsourced image computation tasks in plaintext domain.

Workflow

The proposed system consists of two main phases as follows:

Data Preprocessing: In the Data Preprocessing phase, for the image I, a user prepares ciphertext C through encoding process *Encode(I)* and sends C to the CCP, where computation takes over the encrypted image C. Such an encoding algorithm should be lightweight and support as many image processing algorithms as possible. Hence, the user only needs to encode its image data once, and CCP takes the majority of the computation workload.

Encrypted Image Evaluation: After receiving the encrypted image data, CCP performs image processing algorithms over the ciphertext domain to get the corresponding encrypted results. Meanwhile, the private information of uploaded image data should be protected from the CCP. (After that, the user can decrypt and get image processing results in plaintext.)

Note that in this system architecture, users can get the maximum flexibility and scalability to perform massive image processing tasks. In fact, if a user has to perform part of an image processing task and then upload the encrypted intermediates to CCP, the user's flexibility will be limited. Under this circumstance, a user will have to compute and encrypt different intermediates for various image processing tasks respectively. Nevertheless, even a minor parameter change in processing algorithms will force the user to compute and encrypt the whole image dataset over again.

Design Targets

After building the system model and defining the workflow, we formulate the design targets in constructing a privacy-preserving image processing mechanism in the cloud: The first design target should be functionality, which requires the proposed system to perform image-processing algorithms and generate corresponding results correctly. The second design target should be security, which requires the proposed system to protect the image contents' confidentiality from the CCP while performing the processing algorithms on ciphertext domain. The last design target should be efficiency, which requires the computational complexity and communications complexity between the user and the CCP to be practical.[7] These three design targets are equally important. However, if we must set a priority, the most important should be security. After all, sensitive information leakage can result in severe losses.

Here, we use image feature detection algorithms as a set of case studies to analyze above three design targets.

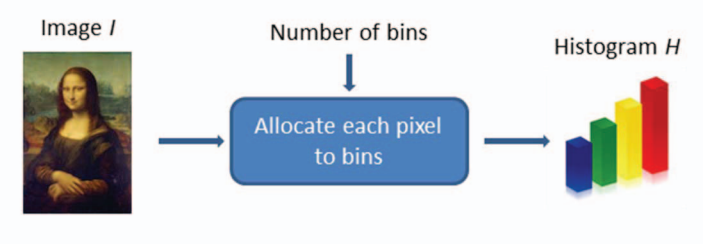

Figure 2. RGB histogram.

Functionality

As we discussed earlier, image feature detection algorithms can be divided into two main categories: global feature detection, e.g., RGB histogram, Color Layout Descriptor (CLD), Color Structure Descriptor (CSD)[6] and so on, and local feature detection, e.g., SIFT, HOG.[15–16] Here, we use the functionality of the RGB histogram as an illustrative example for global feature detection algorithms. In color feature detection algorithms, the histogram descriptor is the most basic descriptor and a building block for advanced feature descriptors. Based on a color histogram, we can compute a series of prevalent color descriptors, including CSD, CLD[15]. As shown in Figure 2, the computation algorithm of a color histogram in plaintext is very simple. However, if we intend to perform this algorithm over the ciphertext domain, the functionality requirement makes it very difficult to be realized by simple encryption schemes: We need to enable the comparison between ciphertext and plaintext to distribute each pixel value into the color histogram correctly. Intuitively, this functionality requirement seems to be contradictive to the design target of security, or the confidentiality of the encrypted image data. If ciphertexts are comparable to plaintexts, the adversary can easily deduce all the values of encrypted pixels and get the sensitive information contained in an image. However, after carefully analyzing the functionality requirement of the histogram algorithm, we find that the exact required functionality is not the result of comparison between ciphertext and plaintext. The required functionality is the corresponding comparison result in the ciphertext domain. Based on this observation, Qin and colleagues utilize a somewhat homomorphic encryption scheme to fulfill the corresponding functional requirements and develop a privacy-preserving image global feature detection algorithm based on it.[6] The corresponding experimental details are described in the paper.

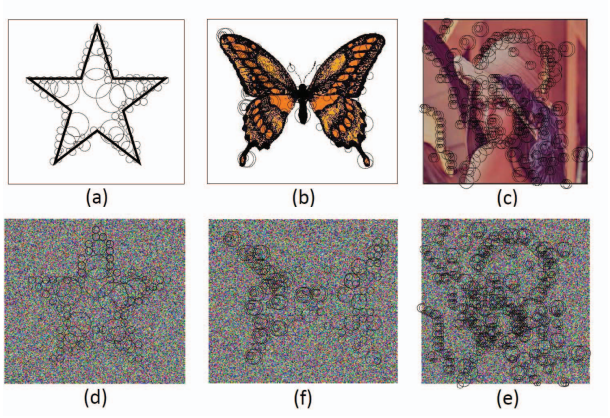

Figure 3. The illustrative experimental result of SIFT Feature Descriptor: Figures (a-c) are the results in the plaintext domain. Figures (d-f) are the corresponding results in the ciphertext domain.

Security

Recall that in the system model described above, we assume the CCP to be honest-but-curious. It means that the CCP will follow the procedures in the protocol and correctly perform the feature detection algorithm over ciphertext domain to protect its credits for the commercial benefits. However, it is still easy for an adversary, e.g., curious cloud engineer, to deduce the sensitive information contained in the image data through monitoring the data flow in ciphertext domain. Specifically, this kind of attack is especially hard to be defended in performing local feature detection algorithms. Here, we use SIFT as an illustrative example from local feature detection algorithms: As a local feature detection algorithm, the SIFT algorithm first needs to detect the location of interesting points in an image. After that, it characterizes interesting points' neighbor pixels by generating the corresponding feature descriptors around it. In Figure 3, circles with different sizes represent different local feature descriptors whose centers are the location of interesting points. In the process of secure image processing, since the CCP needs to generate those local descriptors, it will inevitably deduce the location of those interesting points in the image. However, from Figure 3, we can find that an eavesdropper on the ciphertext domain data flow can easily get rough shapes of objects in the image. Through analysis, we discover that this problem is similar to the pixel value comparison problem we met in color histogram algorithm. Can this problem can be solved by following the same methodology? More specifically, is it possible to solve this problem by encrypting the location of pixels and enabling the detection of interesting points on the ciphertext domain? Unfortunately, this methodology can only convert the problem from the contradiction between security and functionality to the contradiction between functionality and efficiency.[8] In fact, based on the complexity analysis of the corresponding method, it is easy to find that additional computational complexity for hiding pixel positions equals the computational complexity of the brute force attack against the encryption scheme. To achieve the functionality requirements, it seems to be impossible for the proposed system to provide a practical efficiency performance under the traditional definition of data confidentiality in cryptography. To solve this problem, Qin and colleagues introduce a multi-server structure-based mechanism to achieve a balance among the functionality, security, and efficiency simultaneously.[6]

Efficiency

In the complexity analysis of secure cloud computing, we need to analyze the efficiency of the proposed mechanism in three aspects[9]: The computational complexity on both the user and the CCP sides, and the communication complexity between these two parties. In practice, to achieve the flexibility on user's side and scalability on the CCP's side, most existing designs only allocate necessary procedures like encryption and decryption tasks to the user. Consequently, complicated functionalities are required in the corresponding mechanisms. It leads to more complicated encryption algorithms being applied that finally overload the user's computational

complexity. Concerning a few homomorphic encryption algorithms, the corresponding encryption and decryption computation complexity is even larger than the computation complexity of performing the image processing algorithm.[16] In this case, the practice of the corresponding mechanism is neglected. Hence, not only do we need to develop an encryption scheme that can provide the number of homomorphic operations required in the image-processing algorithm, but we also have to carefully balance the computation and communication complexity to ensure the feasibility of the proposed design.[17]

HOMOMORPHIC ENCRYPTION BASED IMAGE PROCESSING

Since the proposal of homomorphic properties, Fully Homomorphic Encryption (FHE) has been considered the Holy Grail in cryptography. After Gentry's breakthrough on lattice-based FHE,[16] a general solution has been shown to allow homomorphic evaluations over the ciphertext domain. However, applying existing general fully homomorphic encryption schemes to image processing applications would be far from practical, due to their huge computation complexity. Different from FHE, SHE schemes can only support limited times of homomorphic operations. Considering the design targets of secure image processing mechanisms, SHE schemes seem to be suitable for some image processing applications. Here, we briefly introduce the framework of the state-of-the-art practical SHE scheme before discussing its merits and drawbacks.

To encrypt the image data, we utilize the most recent Ring Learning with Error (RLWE) based SHE scheme from[17]. Here, we first define SHE scheme to be a tuple of algorithms: *SHE = (SH.Gen, SH.Enc, SH.Add, SH.Mult, SH.Dec)*. Among these algorithms, *SH.Gen* defines the secret key *sk* by sampling a ring element. It randomly generates ring elements a_0, e, and computes the public key $pk = (a_1 = -a_0e+te, a_0)$. The *SH.Enc* and *SH.Dec* are shown below:

$$SH.Enc(pk, m) = ct = (c_0 = a_0u + tg + m, c_1 = a_1u + tf') \quad (1)$$

$$SH.Dec(ct, sk) = m = \sum_{i=0}^{\delta} c_i s^i (\mathrm{mod}\, t) \quad (2)$$

$$SH.Add(ct, ct^*) = (c_0 + c_0^*, \ldots, c_{\max} + c_{\max}^*) \quad (3)$$

$$SH.Mult(ct, ct^*) = (\sum_{i=0}^{\delta} c_i v^i) \times (\sum_{i=0}^{\delta} c_i^* v^i) \quad (4)$$

Note that the above properties are only valid within the finite number of homomorphic additions and multiplications.

The SHE schemes are usually utilized in both secure image feature detection and secure image retrieval matching mechanisms. One typical example is in privacy-preserving face recognition mechanisms.[13] In the corresponding system model, a user first uploads an encrypted facial picture to cloud server as a query. After that, the cloud server performs the feature vector detection operations and decomposes it into multiple Eigen faces over ciphertext domains. These ciphertexts are compared with the database held by the cloud server to find matches through computing their Euclidean distances. Similar feature matching algorithms have been studied and implemented in various biometric matching, classification and clustering algorithms.[18]

However, one opening problem in implementing SHE schemes is the finite number of multiplications. Configuring the scheme to support more multiplication operations leads to impractical computational complexity (increasing several orders of time complexity than the original algorithm). Since the number of multiplications will rapidly increase the computational complexity of homomorphic operations, some existing works try to avoid it by combining secure multiparty computation (SMC) techniques and homomorphic encryption schemes.[19] In these designs, the operations that require high computational complexity in homomorphic encryption schemes, like multiplication and factorial are realized by using SMC based mechanisms instead of homomorphic operations. This design methodology provides better performance on efficiency compared with completely using homomorphic encryption-based solutions.

SMC BASED IMAGE PROCESSING

Usually, the Secure Multiparty Communication[20] protocol is considered as a general solution to any function computations. However, due to its enormous computation and communication complexity, it is not widely implemented in practice. Nevertheless, its advantage in compatibility and the simplicity of the SMC algorithm makes it play a very important role in secure cloud computing mechanism designs. Among many SMC techniques, the Secure Two-party Computation is often utilized as a building block in constructing the system with techniques like homomorphic encryption scheme.

SMC based Secure Image Feature Detection

In image feature detection algorithms, functionality requirements like the comparison, factorial, and trigonometric operations exist in many complicated image feature detection algorithms. However, these operations over the ciphertext domain require tens to hundreds of iterations of homomorphic addition and multiplication operations. Hence, it seems to be impractical to use only homomorphic encryption based techniques to realize all those functionalities. To solve this problem, one possible methodology is adjusting the system architecture of the cloud computing platform to utilize SMC techniques. As shown in Figure 4, a user can easily realize homomorphic additions and ciphertext comparisons through introducing additional cloud servers—for example, a simple implementation of the one-time-pad encryption scheme from SMC protocols that splits one plaintext into two ciphertexts enables homomorphic additions. This design methodology can be generalized to utilize various arithmetic encryption methods from SMC protocols. Moreover, the utilization of the SMC technique also provides an additional perspective to balance the communication complexity and computation complexity. As Figure 4 shows, in the framework of the SMC technique, it is effective to alleviate computation complexity on the user side by introducing additional communication complexity between the user and cloud servers, e.g., through uploading ciphertexts to two cloud servers. Experimental results of diverse applications, refer to the literature.[6–9]

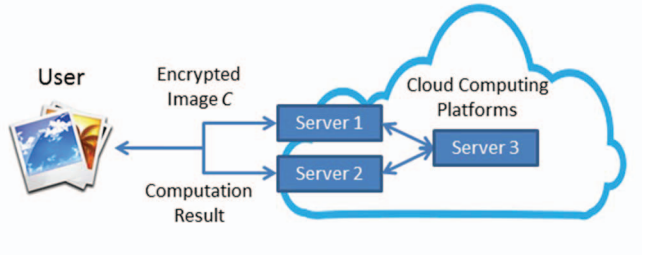

Figure 4. Utilizing SMC techniques by introducing additional cloud computing platform

SMC based Secure Image Digital Watermarking

Another popular implementation of SMC techniques lies in secure image watermark detection algorithms. A digital watermark is a signal permanently embedded in digital data, i.e., audio, pictures, or video. This signal is hidden in the host data in such a way that it is inseparable from the data and so that it is resistant to many operations not degrading the host data. The watermark can be detected or extracted later through computing operations to make assertions about the data. Various secure digital watermarking system models have been proposed. In existing works, the cloud is usually utilized to perform tasks like watermark generation, detection, and matching. A typical model of watermark detection is shown in Figure 5.

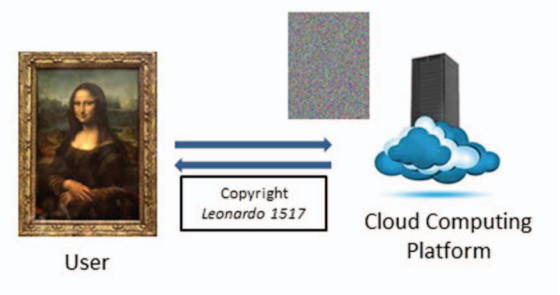

Figure 5. Workflow of secure digital watermarking detection in the cloud.

To construct a secure watermark detection mechanism, most existing solutions leverage SMC techniques. Malkin and colleagues authors propose to use both secure sharing and watermarking schemes to protect users' media data in the cloud.[21] Similar to the multiserver structure utilized in image feature detection applications, the proposed secure sharing scheme divides users' data into multiple pieces and uploads it to different cloud servers, making it difficult to derive the entire information from any single cloud. Lin and colleagues focus on the efficiency of video data watermarking. Their work splits the original video and uses a Hadoop distributed computing system for the different requirements to realize watermark embedding.[22] Diephuis and colleagues propose a framework for message privacy-preserving copy detection and watermark identification based on the signs of the Discrete Cosine Transform (DCT) coefficients.[23] The architecture allows for searching encrypted data and places the computational overhead on a cloud server. The low-frequency DCT coefficients are utilized to generate a dual set of keys to encrypt the source image and a robust hash for the digital watermarking queries.

Moreover, by utilizing SMC technique, some secure watermarking tasks performed on CCP side have shown close performance as performed in the plaintext domain. Using random matrix transformation, which can be considered as a two-dimensional extension of the one-time-pad technique, an efficient privacy-preserving watermarking detection mechanism is proposed in the work of Wang and colleagues.[24] In a nutshell, using SMC techniques, privacy-preserving watermarking processing mechanisms are secure and efficient under certain conditions.

However, though utilizing SMC techniques can effectively reduce computational complexity compared with using HE techniques, the inherent feature of SMC techniques requires that the data owner must be online when the cloud performs most operations. Hence, it only applies to limited types of applications in practice.

CONCLUSION AND FUTURE WORK

This article studies the problem of privacy-preserving image processing in the cloud, which could enable robust image-processing based applications on devices with limited computation power, e.g., a variety of instant image processing apps on lenses, watches, or other personal devices. Compared with other outsourced computation tasks, image-processing algorithms are relatively complicated and have high computation complexity. To solve the problem, we start by building a system model and formulating design targets. After that, state-of-the-art techniques are introduced, including homomorphic encryption, secure multiparty computation, and so on. We also present several case studies for different techniques and analyze their merits and drawbacks. Through the analysis, we find that the balance among design targets: functionality, security, and efficiency makes it difficult to solve the problem by applying only one technique. The integration of different techniques instead of traditional cryptography tools is the most promising research direction in this area. Also, considering the prevalence of JPEG compression among some data, privacy-preserving decompression of JPEG file as a special case of privacy-preserving DCT computation is also a promising research direction in this area.

ACKNOWLEDGMENT

This work was supported by the US National Science Foundation under grants CNS-1262277.

REFERENCES

1. M. Armbrust et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010, pp. 50–58.
2. H. Esfahani et al., "Cloudbuild: Microsoft's Distributed and Caching Build Service," *Software Engineering in Practice* (SEIP 16), 2016.
3. C. Wang et al., "Privacy-assured outsourcing of image reconstruction service in cloud," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, 2013, pp. 166–177.
4. C. Modi et al., "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, 2013, pp. 42–57.
5. W. Lu et al., "Secure image retrieval through feature protection," *Procedings of the International Conference on Acoustics, Speech, and Signal Processing* (ICASSP 09), 2009.
6. Z. Qin et al., "Privacy-preserving outsourcing of image global feature detection," *Proceedings of the Global Communications Conference* (GLOBECOM 14), 2014.
7. C.-Y. Hsu et al., "Image feature extraction in encrypted domain with privacy-preserving SIFT," *IEEE Transactions on Image Processing*, vol. 21, no. 11, 2012, pp. 4593–4607.
8. C.-Y. Hsu et al., "Homomorphic encryption-based secure SIFT for privacy-preserving feature extraction," *Proceedings of SPIE* (SPIE 11), 2011.
9. Z. Qin et al., "Towards efficient privacy-preserving image feature ex-traction in cloud computing," *Proceedings of the 2014 ACM on Multimedia Conference* (MM 14), 2014.
10. J. Eggers, J. Su, and B. Girod, "Public key watermarking by eigenvectors of linear transforms," *Proceedings of the European Symposium on Security and Privacy* (Euro SP), 2000.
11. H. Wang et al., "Security protection between users and the mobile media cloud," *IEEE Communications Magazine*, 2014.
12. W. Lu et al., "Enabling search over encrypted multimedia databases," *Proceedings of SPIE* (SPIE), 2009.
13. Z. Erkin et al., "Privacy-preserving face recognition," *Proceedings of Privacy Enhancing Technologies Symposium* (PETS 09), 2009.
14. K. Ivanova et al., "Features for art painting classification based on vector quantization of mpeg-7 descriptors," *Data Engineering and Management*, Springer, 2012.
15. T. Sikor, "The MPEG-7 visual standard for content description-an overview," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 6, 2001, pp. 696–702.
16. C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proceedings of the 41st Annual ACM Symposium on Theory of Computing* (STOC 09), 2009.
17. M. Naehrig et al., "Can homomorphic encryption be practical?," *Proceedings of ACM Cloud Computing Security Workshop* (CCSW 11), 2011.
18. M.K. Khan, J. Zhang, and K. Alghathbar, "Challenge-response-based biometric image scrambling for secure personal identification," *Future Generation Computer Systems*, vol. 27, no. 4, 2011, pp. 411–418.
19. S. Pandey et al., "An autonomic cloud environment for hosting ECG data analysis services," *Future Generation Computer Systems*, vol. 28, no. 1, 2012, pp. 147–154.
20. O. Goldreich, *Secure multi-party computation* Manuscript, 1998.
21. M. Malkin and T. Kalker, "A cryptographic method for secure watermark detection," *Proceedings of the 8th International Workshop on Information Hiding*, 2006.
22. C. Lin, C. Lee, and S. Chien, "Digital Video Watermarking on Cloud Computing Environments," *Proceedings of the Second International Conference on Cyber Security* (CyberSec 13), 2013.

23. M. Diephuis et al., "DCT sign based robust privacy preserving image copy detection for cloud-based systems," *Proceedings of Content Based Multimedia Indexing* (CBMI 12), 2012.
24. Q. Wang, W. Zeng, and J. Tian, "A compressive sensing based secure watermark detection and privacy preserving storage framework," *IEEE Transactions on Image Processing*, vol. 23, no. 3, 2014, pp. 1317–1328.

ABOUT THE AUTHORS

Zhan Qin received a BS degree from Beijing Institute of Technology, an MS degree from Columbia University and a PhD from the State University of New York at Buffalo in 2010, 2012, and 2017. He is currently an assistant professor in the department of electrical and computer engineering at the University of Texas at San Antonio. His research interests are in the areas of cloud computing and security with a focus on multimedia data, differential privacy data collection and publication, and cybersecurity in the smart grid. He is a member of the IEEE and ACM. Contact him at zhan.qin@utsa.edu.

Jian Weng received his MS. and BS degrees in computer science and engineering from South China University of Technology, in 2004 and 2000, respectively, and his Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, in 2008. From April 2008 to March 2010, he was a postdoc in the School of Information Systems, Singapore Management University. Currently, he is a professor and executive dean with the School of Information Technology, Jinan University. He served as PC co-chair or PC member for more than 20 international conferences. He won the 2014 cryptographic innovation award from the Chinese Association for Cryptographic Research, the best paper award from SCIS 2011, and the best student award from ProvSec 2014. Contact him at cryptjweng@gmail.com.

Yong Cui received a BE degree and a PhD from Tsinghua University. He is currently a full professor at Tsinghua University. He received the National Award for Technological Invention in 2013, the Influential Invention Award from the China Information Industry in 2012 and 2004, and several Best Paper Awards in refereed international conferences. He co-authored seven Internet standard documents for his proposal on IPv6 transition technologies. His major research interests include mobile cloud computing and the next-generation Internet. Contact him at cuiyong@tsinghua.edu.cn.

Kui Ren received a Ph.D. from the Worcester Polytechnic Institute. He is currently an associate professor of Computer Science and Engineering and the Director of the UbiSeC Laboratory at the University at Buffalo. His research has been supported by NSF, DoE, AFRL, MSR, and Amazon. He has authored 200 peer-reviewed journal and conference papers. His current research interest spans cloud and outsourcing security, wireless and wearable system security, and human-centered computing. He is an IEEE fellow and a Distinguished Lecturer of the IEEE Vehicular Technology Society. He was a recipient of the NSF CAREER Award in 2011 and the Sigma Xi/IIT Research Excellence Award in 2012. Contact him at kuiren@buffalo.edu.

An Empirical Study of Cloud API Issues

Zhongwei Li and Qinghua Lu
China University of Petroleum

Liming Zhu and Xiwei Xu
Commonwealth Scientific and Industrial Research Organisation

Yue Liu and Weishan Zhang
China University of Petroleum

With the emergence of the DevOps movement, software engineers are starting to rely on cloud platform APIs for implementing many fault tolerance, self-adaptation, and continuous delivery features such as deployment changes, scaling out/in, exception handling, backup/recovery, and migration. In this article, we present an empirical study of API issues in commercial cloud platforms. We classify the API failures and their causes, and discuss possible remedies for each category to improve the reliability of cloud applications and introduce our solutions to deal with resource characteristics faults and late timing failures.

The DevOps movement in industry is giving software engineers more responsibilities in terms of using operation environment/platform (for example, Cloud) facilities to automate the deployment, configuration, and run-time environment error detection to enable better self-adaptation and fault tolerance of the application. Such deployment of an application into the cloud requires special programs that handle configurations and provisioning that are specific to the cloud platforms. Once the application is running and performing its normal activities, these programs, sometimes in collaboration with the application, heavily rely on cloud infrastructure APIs to perform sporadic activities such as deployment change, scaling out/in, fault tolerance, backup/recovery, and migration. Some operations are performed through a management console or scripts, but they all use cloud platform APIs behind the scenes to complete the operations. As cloud consumers have limited visibility and control of the public cloud infrastructure, cloud platform API calls are sometimes the only interaction points between the cloud infrastructure and the cloud applications.

Various online discussion forums and our own cloud product development experiences (www.yuruware.com) highlight many issues with these APIs especially when they are needed for reacting to rapid changes in the environment (for instance, outages or workload spikes) or dealing with error-prone operational processes (for instance, upgrade, backup, and recovery).

Copublished by the IEEE CS and IEEE ComSoc
2325-6095/18/$33.00 USD ©2018 IEEE

Many cloud applications are architected to achieve high availability through on-demand resource allocation, micro rebooting, replication, and failover across geographically distributed sites (techblog.netflix.com, www.yuruware.com). Most of these operations are also performed through API calls and often under stress conditions. The application availability depends on the availability of cloud resources and the reliability and performance of these API calls. Software engineers need to understand the nature of the potential faults with API calls and implement proper failure handling and fault tolerance mechanisms for dealing both with API faults and the underlying resources faults.

This work started as a small empirical study[1] of 5 Amazon Elastic Compute Cloud (EC2; aws.amazon.com/ec2) calls out of our own frustration in using them to develop a cloud management software. Cloud management software relies heavily on cloud APIs to perform operations such as snapshotting/starting/stopping VMs, attaching/detaching volumes to a VM to do analysis, and redeploying a system with properly adjusted security, network, and topology settings. The motivation of the study also came from the fact that a large percentage (53%) of the cases reported in the Amazon EC2 forum are related to API failures. In the study of Qinghua Lu and colleagues,[1] we searched only the EC2 forums for 5 highly used calls. We identified 922 cases and classified those using traditional failure and fault categories. One of our conclusions was that the traditional classification did not provide many insights into the cloud context and possible remedies.

In this article, we report on a broader empirical study of Cloud infrastructure API issues using publically available information. The sources include 32 cloud platforms' discussion forums (32 cloud platforms that provide similar APIs to Amazon EC2 to supplement the main findings), technical analysis of API issues during outages from reputable sources, such as Amazon outage reports (aws.amazon.com/message/65648/, aws.amazon.com/message/67457/, aws.amazon.com/message/65649/, aws.amazon.com/message/2329B7/, aws.amazon.com/message/680342/), Netflix technical blogs (techblog.netflix.com), and AvailabilityDigest (www.availabilitydigest.com) and our own experience on product development on Cloud (www.yuruware.com).

The major contributions of this article include the following:

- We extracted 2087 API failures from a wide range of sources (the 922 cases in our previous work are included).
- We classified the API failures using the traditional failure classification[2] in dependable computing. In addition, we discussed the error messages and different potential remedies for each category.
- We classified the faults (causes of these failures) using a new scheme. We have some initial recommendations targeted for each category.
- We discussed research needs, tooling needs, and some potential design remedies that can be used by cloud application designers.

METHODOLOGY

Study Sources

The study sources include API discussion forums and technical analysis of API issues during outages from reputable sources (such as Amazon outage reports, Netflix Tech Blog, AvailabilityDigest), which are all publically available information.

Our study considers 32 platforms (the first four are major sources): Amazon EC2, Abiquo, Bluebox, Brightbox, CloudSigma, CloudStack, Dreamhost, Enomaly, ElasticHosts, Eucalyptus, Gandi.net, GoGrid, Google Compute Engine, Joyent, IBM SCE, Linode, Nimbus, Ninefold, OpenNebula, OpenStack, OpenSource Cloud, Rackspace, RimuHosting, Slicehost, SoftLayer, Terremark, VMware vCloud, VCL cloud, Voxel, VPS.net, skalicloud, serverlove.

In addition to the above publicly available information, during the development and operations of our commercial cloud management product (www.yuruware.com), we learned many lessons with respect to EC2 API reliability and performance.

Study Procedures

Data Collection

As cloud storage issues are often very different from the cloud compute issues, the scope of this study is on cloud compute related operations such as describe, reboot, create, destroy, make snapshot/images virtual machines or nodes, and attach/detach volumes to an instance. We also cover some network related issues including IP management and load balancer management such as attach/release IP addresses from an instance, and create, describe, attach, and detach load balancers to a subnet and their configuration as these are often critical during outages.

Our study scope started January 2011; the evolution of the platforms may make some of the old cases irrelevant. We collected 2,087 failures and 1,466 faults in total. The majority of them come from Amazon EC2 (1,846 failures and 1,326 faults).

Failure and Fault Classification

As shown in Figure 1, we largely classified the API failures and faults using the taxonomy of dependable and secure computing[2] with important improvements making them more cloud specific. The percentage distribution of failure and fault categories gives some insights into the severity and prevalence of the issues while the categories themselves give inspiration on category-specific solutions especially from a cloud consumer perspective.

A failure is an event that occurs when the delivered service deviates from the correct service. We consider all failed or slow API calls as failures. We documented the symptoms of the failures and, if available, the error messages.

A fault is the adjudged or hypothesized cause of an error.[2] In this study, we try to locate the causes of a manifested failure. Not all failure cases have a cause discussion in the original thread. However, many failures are very similar to each other and happen in similar contexts. For example, a large number of API call failures were reported immediately after an outage. The final outage analysis reports then go into details discussing the causes of these API call failures. Some failure discussions point to other threads that may have a fault discussion. As we have analyzed a very large sample, we are in a unique position to link similar failure issues and reuse faults discussions in some of these linked failures to represent all of them. The number of fault cases is smaller than the number of failure cases as we could not locate some of failure causes.

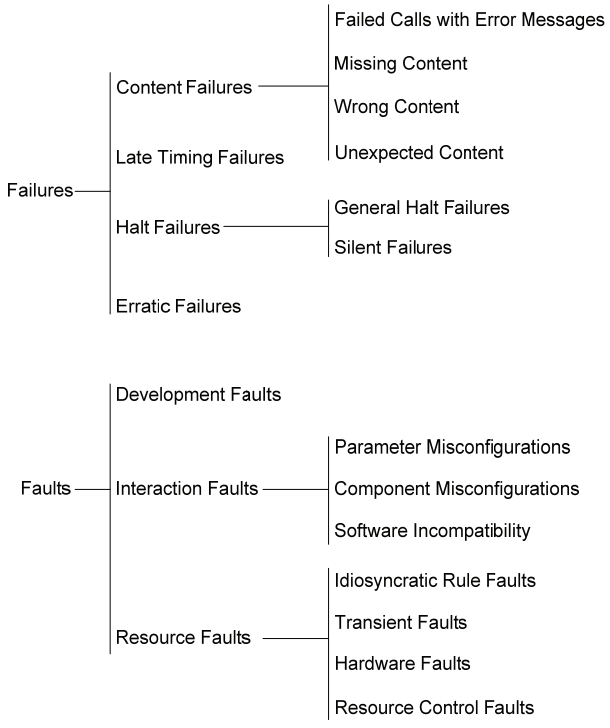

Figure 1. Classifications of failures and faults.

CLASSIFICATION OF API FAILURES

We classify the reported API failures into four major types: content failures, late timing failures, halt failures, and erratic failures. Among reported API failures, 55% are content failures, 35% are halt failures, 6% are late timing failures, and 4% are erratic failures.

Content Failures

Content failure is defined[2] as "the content of the information delivered at the service interface deviates from implementing the system function." We further classify content failures into four sub-types:

- failed calls with error messages (49%),
- missing content (2%),
- wrong content (2%), and
- unexpected content (2%).

Failed Calls with Error Messages

Only 49% of the overall API failures are failed calls with error messages. In these cases, 58% of the time, users understood the problem from the error message but did not know the right solutions to the problem, while 42% of the time, users could not pinpoint the problems from the error message.

Symptom: When a user tried to start an instance, the operation failed with an unclear error message.

Root cause: Unknown.

Solution: AWS engineers advised detaching the EBS volume from the instance and attaching it to another running instance.

Error message: State Transition Reason - Server.InternalError: Internal error on launch

In the above case, the user could not pinpoint the issue from an unclear error message. The failure could be caused by using the wrong name for the root volume. The root volume must be named /dev/sda1. Many people used /dev/sda instead, and it may generate the Server.InternalError message upon launch. Intuitively, the infrastructure software could know the problem precisely and generate a more specific error message. From a cloud consumer perspective, diagnosis or configuration checking tools using a knowledge base could have encoded the specific rule and helped diagnosis.

Symptom: Failed API calls with Request limit exceeded error message.

Root cause: API calls exceeded limit.

Solution: N/A. There is no official information on the limit or the time span on which the limit is calculated or suggested wait time.

Error message: Client.RequestLimitExceeded: Request limit exceeded

The returned error message in the case above pinpoints the problem. However, the users could not take specific actions. This is well known and studied in the bug community when an error message is helpful only to internal developers rather than developers who are interacting with the software. This is also true in misconfiguration studies[3] where people cannot act on a message. Or in the worst-case scenario, a specific error message misleads the user/developer to the wrong place.

Missing Content

This type of failure captures the cases where some information is missing in the returned output after an API call. Two percent of the cases are in this category, and most of them are related to "describe" type of calls, for example, ec2-describe-instances, which are used for monitoring.

Symptom: A user tried "ec2-describe-instances --filter "platform="" using the EC2 API tools and received nothing when he has some running Linux instances. In the API document, it says "Use windows if you have Windows based instances; otherwise, leave blank."

Root cause: API bug.

Solution: The AWS engineer forwarded this to the EC2 development team to investigate.

The case above shows an API failure case where the content is missing in the returned list of instances. A bug caused this particular case. The EC2 development team later found there was no easy way to filter Linux instances. The users either do not use the "platform" filter and it will return all instances or only use the windows filter. Despite the difficulties in reliably identifying Linux instances and the bug, returning the total number of running instances alongside of the empty filtered list could have helped the users diagnose the failure faster. Another popular case is related to transient missing information perhaps due to internal information propagation and eventual consistency. Cloud platforms should inform consumers that certain information might not show up immediately and give an estimated time for eventual consistency.

Wrong Content

Two percent of cases report that the output is wrong or the output is inconsistent from different calls. This usually happens when a user tries to detach a volume or describe an instance or a volume.

> **Symptom:** The output of ValidFrom and ValidUntil are swapped in ec2-describe-spot-instances in ec2-api-tools 1.6.1.4.
>
> **Root cause:** API bug.
>
> **Solution:** Use defensive programming to correct the errors.

In the above case, an AWS engineer recommended using the latest version of the EC2 API tool "ec2-api-tools 1.6.1.4." The user came back and reported that the latest version had the same issue, but AWS did not follow up with the case.

> **Symptom:** A user found that the output of ec2-describe-instances and ec2-describe-volumes did not match: the output of ec2-describe-instances showed the volume was attached while the output of ec2-describe-volumes showed the volume was attaching.
>
> **Root cause:** N/A
>
> **Solution:** Wait.

The case above shows that the outputs of two different API calls about the same information are inconsistent, which happens very frequently. In some cases, it is a transient problem and the answers will converge. In other cases, the inconsistency remains. This depends on the internal implementation of the APIs and how the state information is stored. For key operations, the cloud consumers cannot trust a single source of information, and in this case, the users have to SSH into the instance to find out the true state.

Unexpected Content

Two percent of the cases report the outputs of the calls are different from the users' expectation. Most of them are related to API calls describing the status of resources.

> **Symptom:** A user launched an instance in the us-west-1 region. When the user ran the ec2-describe-instances command, it shows that the instance did not exist. When the user listed all instances, the output showed all of instances in us-east-1 region.
>
> **Root cause:** Misconfiguration.
>
> **Solution:** Using the --region option to specify us-west-1.

The common failure shown above is caused by user misconfiguration. Many users thought the ec2-describe-instance command returns instances in all EC2 regions, and they spent a significant amount of time diagnosing expectation mismatch. This is not technically a failure. However, the returned results could be more informative by stating that the current results contain only instances from a specific region. To have the original intentions of the call be part of the self-describing information of the returned results (rather than implicitly assumed) can help users diagnose issues.

Late Timing Failures

Late timing failure means that the arrival time of the delivered information at the interface deviates from implementing the system function,[2] but it does eventually arrive. Six percent of cases featured complaints about slow responses to users' API calls. As explained by the AWS support team, the slowest API calls can take several minutes to complete.

> **Symptom:** It took 16 minutes for an instance to stop.
>
> **Root cause:** N/A.
>
> **Solution:** The AWS engineer advised to try "force stop" twice if this happens next time.

The above case shows a timing failure. Many users experience longer time to complete, for example, half hour or even a couple of hours. However, in the API document, it does not provide information on the normal time to complete each API call. This is understandable for some calls such as starting/stopping/snapshotting an instance, where the time highly depends on the characteristics of the instance. However, some information about typical instances could help. For other API calls such as attaching/detaching volumes or regarding load balancers or IP management, a general expectation on timing and recommended options for resolving late timing failures could be very helpful.

In large-scale distributed systems, latency variability is something that cannot be avoided but only tolerated. For example, Netflix Hystrix builds a timing profile on services and calls so that it can choose to fail fast after the waiting time has passed a certain percentile (for instance, 95%). We also built EC2 API call timing profiles during the development of our commercial product. We measured the 5 most frequent EC2 API calls ("launch instance," "start instance," "stop instance," "attach volume," and "detach volume") by calling the API 1,000 times and recording the return time under various realistic conditions.[5]

Halt Failures

Halt failures refer to cases where the external state becomes constant.[2] API calls in halt failures do not return while API calls in late timing failures do return after some time. A special case of halt failures is silent failures, which means no delivered service at all at the system interface. Thirty-five percent of the failures were in this category.

General Halt Failures

Table 1. General Halt Failures.

Intended call	Halted state	Percentage
Stop an instance	Stopping	63%
Start an instance	Initializing or pending	6%
Detach a volume	Detaching	31%

The most frequent API issue is that an API call is stuck at a certain state. Table 1 shows the percentage distribution of API calls stuck at different states.

> **Symptom:** A user reported that the instance is stuck at stopping and "force stop" would not help.
>
> **Root cause:** N/A.
>
> **Solution:** The AWS engineer stopped the instance for the user on the AWS side.

Side effect: AWS killed the volume that the user was hoping to reuse.

Many users experience calls with stuck states subsequently. One example is shown above. It often happens that the AWS support team helps the user stop an instance at the AWS side, and the user comes back to complain that the instance is hanging at "initializing" or "pending" state when the user tries to start the instance.

On Amazon EC2, stopping/re-starting an instance changes the underlying physical machine for the instance. Users often try to stop and re-start instances after they cannot connect to their instances. After stopping the unresponsive instance and re-starting the instance, the instance can move from the unhealthy host to a healthy machine. If users are unable to stop their instances, they often try to detach their volumes and attach them to other instances.

Messages around stuck calls are generally non-existent and users can do very little. Experienced users have good snapshots and usually do not wait in such cases, as the possibility of recovery even with AWS assistance is very small.

Silent Failures

The second most frequent API failure category is unresponsive calls, which accounts for 18% of the cases.

Symptom: An instance was not accessible and the user could not stop/start it or creates a snapshot of it.

Root cause: AWS outage.

Solution: The AWS engineer advised that the user must launch a replacement instance from a pre-existing backup (EBS AMI). Attempts to stop an inaccessible instance will likely result in an instance becoming stuck in the stopping state. Customers that do not have a known good backup must wait for the issue to be resolved for their instance connectivity to be restored.

The case above shows a silent failure caused by an AWS outage. During API call throttling or an outage, the platform should still try to return meaningful messages over API calls rather than fail silently or, worse, carry out the calls without telling the caller.

Erratic Failures

Erratic failures refer to the case when the delivered service is unpredictable.[2] Four percent of cases report erratic failures. This type of call includes two subtypes: 1) the call was pending for a certain time and then returned to the original state, or 2) the call was successfully executed first but failed eventually.

Symptom: A user tried to start the instance several times. It indicated that the status is pending and it goes back to stop.

Root cause: N/A.

Solution: The AWS engineer returned the user's EBS volume to the available state and believed this would resolve the user's problem.

The first sub-type occurs when a user tries to start an instance, which takes 68% of erratic failures. One example is shown above, which was due to the EBS volume issue.

Symptom: A user associated an elastic IP with an instance and could SSH into the instance with the elastic IP. After a few minutes, the elastic IP was silently disassociated from the instance.

Root cause: An issue with the underlying host.

Solution: The AWS engineer advised that the quickest fix was to stop and then start the instance to relocate to a different host.

As shown above, the second sub-type of erratic failures happens to elastic IP association, which is 32% of erratic failures. Cloud consumers cannot completely trust an API call telling you a particular operation has been successfully completed. But it only happens to some specific operations rather than broadly. Cloud consumers should consider a delayed check of status on small calls independently before proceeding confidently to subsequent operations.

FAULTS (CAUSES OF API FAILURES)

The classification proposed in the research of Avizienis and colleagues[2] includes three categories, including development faults, interaction faults, and hardware faults. In the context of Cloud API, development faults concern bugs inside the API implementation or third-party software residing in the cloud platform, not from consumer applications. Interaction faults refer to the ones caused by misconfigurations by API users. We further classify the interaction faults into parameter misconfiguration, component misconfiguration, and software incompatibility misconfiguration.[3] The distribution of misconfiguration faults in our study largely matches the distribution in the report by Z. Yin and colleagues.[3] We proposed resource faults to replace the hardware faults and include hardware faults as a sub-category to suit the cloud computing setting.

Development Faults

Software bugs are a minor cause of API failures. In the reported faults, 2% are due to bugs and 83% come from API implementation bugs. Other bugs are from third-party software residing in the cloud platform. We do not count bugs inside users' applications. Here are several representatives of API bugs confirmed by the development team:

output format of ec2-describe-resereved-instances is messed up

the volume states in the outputs of ec2-describe-instances and ec2-describe-volumes do not match

as-describe-auto-scaling-groups reports wrong launch configuration name for instances which are in 'Terminating' state

the output of --valid-from, --valid-until swapped in ec2-describe-spot-instances

ec2-describe-instances returns an extra undocumented column after the Kernel ID column

ec2-describe-spot-price-history does not report spot price for Linux m1.large instance type

For most of the API bugs reported by the consumers, cloud vendors fix the reported bugs and release a new API version. However, the correction process normally takes several weeks.

Interaction Faults

Misconfiguration is the most frequent cause (32%) of the reported API failures. We went through the 33% API failures caused by misconfiguration faults and reclassified misconfiguration faults into three types: parameter misconfiguration, component misconfiguration, and software incompatibility misconfiguration.[3]

Parameter Misconfiguration

The most frequent misconfiguration fault type is parameter misconfiguration, which is 75% of all misconfiguration faults. In the category of parameter misconfiguration faults, we further classified the reported cases into three types: legal but undesired (18%), illegal value—environment inconsistency (58%), and illegal value—value inconsistency (24%).

Legal but undesired faults refer to the faults that have "perfectly legal parameters but do not deliver the functionality intended by users." The case below shows a frequently occurring legal but undesired fault. There are a large number of cases in the EC2 forum discussing the API issues due to the misconfiguration of the parameter "--region REGION" regarding the --region option and setting of the EC2_URL environment variable. The API Tool Reference (docs.aws.amazon.com/AWSEC2/latest/CommandLineReference/CLTRG-common-args-api.html) does not mention how to set EC2_URL other than us-east-1, although it's easy to guess.

> **Symptom:** When the user runs ec2-describe-spot-price-history, the user only gets results for us-east.
>
> **Root cause:** Misconfiguration: the user should specify –region option.

Illegal value faults refer to the faults that have correct parameter format but violated value constraints. Illegal value faults include environment inconsistency faults and value inconsistency faults.[3] Environment inconsistency faults refer to when "some parameter's setting is inconsistent with the system environment." The case below shows an environment inconsistency fault example.

> **Symptom:** The user executed ec2-describe-images and received a message saying "The system cannot find the path specified."
>
> **Root cause:** Misconfiguration: The user set the EC2_HOME path to a wrong path.

Value inconsistency faults refer to when "some parameter settings violate some relationship constraints with some other parameters." Below shows a value inconsistency fault example.

> **Symptom:** When the user tried to start an instance, the instance went to "pending" status, and then went back to "stopped" status.
>
> **Root cause:** Misconfiguration: The user had tried to attach an EBS volume to xvdq. However, Windows instances cannot have a block device greater than xvdq, which is configured by AWS.

The symptoms and feedback messages around the above two examples can also be improved by incorporating the obviously known and specific cause.

Component Misconfiguration

Twenty percent of misconfiguration faults are component misconfigurations. These cases occur when a component is missing, and they are mainly about misconfiguration in security group settings. Below gives an example of such type of fault.

Symptom: The user was unable to start the instance and getting the error saying "Instance does not have a volume attached at root."

Root cause: There was an inconsistency between the root device configuration and the changed environment.

Software Incompatibility

Five percent of misconfiguration faults are software incompatibility faults, an example of which is shown below.

Symptom: The user was unable to use ec2-describe-regions.

Root cause: Misconfiguration: The user did not install the right version of standard J2SE.

Users usually are not able to receive meaningful feedback from API failures regarding interaction faults. This mirrors the results from Yin when analyzing configuration errors.[3] When an API failure happens, the returned error message should clearly explain failures and faults and advise how to fix them. When a user tries to do some important operations, to avoid misconfigurations, an alert message could explain the outcomes led by the operation and ask the users to confirm the operation. On the one hand, the idea that developers should provide clear error messages when errors are detected is decades old. On the other hand, as our results and results from Yin show, this idea is not universally adopted. An open issue is how to enforce such a requirement.

Resource Faults

A significant percentage of faults (66%) in our study are related to characteristics of the resource being operated on by the API calls. This is significantly different from traditional operator errors and misconfigurations. API calls are essentially operating on coarse-grained resources (virtual machines, volumes, IP addresses, and load balancers). We classify resource faults into four new categories: idiosyncratic rule faults (2%), transient faults (14%), hardware faults (28%), and resource control faults (56%).

Idiosyncratic Rule Faults

Idiosyncratic rule faults refer to the faults caused by violating the idiosyncratic rules that a resource has. These rules are not transient in nature and not related to simple configurations. Here are a few examples:

Windows instances can only have 16 volumes attached and device names only range up to xvdq.

The IDs for the same virtual machine image are different across regions. When users make API calls in a new region, using the wrong ID often causes a failure.

Without rebooting, a Linux instance may not recycle its mount points. If you have mounted more than 24 devices, an API call for attaching a new volume may fail if you try to use the same device name used before even if it is not empty.

A volume cannot be detached from an instance due to corrupted information around processes still accessing the volume.

The error messages for the above failures are often very unclear. The API/system knows why the failures happen but did not communicate that information back to the user. The error message returned by API should be specific.

Transient Faults

Transient faults usually cause time variability of API calls or temporary inconsistency among different calls requesting the same information. Some representative examples are:

> Registered instances fail load balancer health check due to timeout but come back later.
>
> There may be a delay (up to 15 minutes) in propagating the SSL certificate to all the regions when the elastic load balancer is created using the AWS management console.
>
> The instance would still appear in the "running" state for some time after it had been terminated.

Normally, the consumer's reaction to this type of faults is to wait and/or retry. However, the waiting time could be different depending on a specific type of transient faults. For example, waiting for SSL certificate to propagate is about 15 minutes, and it may imply a different type of fault after that. Other waiting time could be based on the 95% percentile of the timing profile of an API call.

For tolerating such faults, there is a large body of work on fault tolerance techniques, self-healing mechanisms, and run-time adaptation. However, API failures in cloud are different in nature. API calls in cloud are often requesting different resources to perform tasks on themselves such as starting/stopping oneself and attaching/detaching a volume to oneself. This is different from requesting a resource to do some application-level computation, which is usually covered by the application code rather than infrastructure API calls. There is a possibility that the requesting process produces failures (API implementation or users' API call code) or, more worth noting, the underlying resources performing the tasks produce failures.

Essentially, calling cloud APIs can be seen as offering work items (jobs) to different cloud resources with different design-time and run-time characteristics. Dealing with failures can be seen as exception handling and job (re)offering mechanisms reacting to failures during an operation achieved through a sequence of API calls. There is a body of literature in the workflow community dealing with such issues.[4-5]

Hardware Faults

A significant portion of failures is caused by hardware-related resource faults within the infrastructure. The most common one is degraded underlying hardware upon which the instances reside. Amazon usually sends an email to the owner of the instance notifying the scheduled date of retirement. The common solution to hardware failures is to move the instance to a new physical machine by stopping and starting the instance.

However, users often experience failures when they try to stop the instance and detach the volume due to the failed or degraded underlying hardware. The AWS support team normally advises them to try force stopping or force detachment more than twice to make the operations work. According to comments in API Tool Reference, both force stopping and force detachment operations may cause serious negative impact, like data loss or a corrupted file system, as the instance will not be able to flush file system caches or file system metadata. After either of these two operations, the user must conduct file system check and repair procedures. This option is not recommended for Windows instances.

Many users even fail with several tries of force stopping or force detachment. The AWS support team has to help users fix the issues on AWS side. Therefore, it would be better for the user to

immediately stop all services running on the instance with degraded hardware or at least stop accepting new jobs once they receive the notification email.

Resource Control Faults

One important category of resource-related faults is not due to the resources themselves but the control of the resources. This means the resources themselves are healthy but the control planes for starting/stopping resources, redirecting traffic to appropriate resources, or health checking could be the problem causing an API call to fail. This happens in almost all outages.

Twenty percent of the reported API failures in the EC2 forum are part of the Amazon outage events. After each Amazon outage event, Amazon usually publishes a summary of the event and analyzes the reasons for that. There are several signs to the user when AWS outage events happen, for example, stuck API call, timeout API call, and degraded performance. AWS outages can be initially caused by software bugs, natural disasters, network overload, improper hardware upgrade, and power outage but often lead to resource control faults later causing a wider range of failures.

REMEDIES

We have interleaved our recommendations throughout the early sections. We summarize them in the following Table 2 and Table 3. We developed our own solutions to tolerate resource characteristic and configuration faults and late timing failure during operations on cloud applications, namely, POD framework[4] and an API wrapper of AWS API.[5]

Table 2. Recommended Solutions to Each Failure Type.

Failure Classification	Some Recommendations for Cloud Users
Content Failure: Failed Calls with Error Messages	Cloud users can use better knowledge-based diagnosis and configuration checking tools pinpointing the causes of failures.
Content Failure: Missing Content	Cloud users should consider relying on multiple sources of answers for critical operations.
Content Failure: Wrong Content	Cloud users should wait, retry, or compare the answers from multiple sources.
Content Failure: Unexpected Content	Cloud users should perform semantic checks on returned answers.
Late Timing Failure	Cloud users should consider building their own fail-fast framework, build timing profiles on APIs, and use that to fail fast and seek alternatives.
Halt Failure	Cloud users should consider more active backup and immediately make resources into passive/quiescent state when encountering issues and seek alternative means rather than wait.
Erratic Failure	Cloud users should consider a delayed check of status on small calls independently before proceeding confidently to subsequent operations.

Table 3. Recommended Solutions to Each Fault Type.

Faults Classification	Some Recommendations for Cloud Users
Development Fault	Cloud users should treat the answers of API calls as unreliable, perform semantic checks of reasonableness, or compare with answers of other calls.
Interaction Fault: Parameter Misconfiguration	Tools that checks configuration errors are needed.
Interaction Fault: Component Misconfiguration	Tools that checks configuration errors are needed.
Interaction Fault: Software Incompatibility	Tools that checks configuration errors are needed.
Resource Fault: Idiosyncratic Rules	Cloud users can use a knowledge-based configuration checking tool to help.
Resource Fault: Transient Faults	Cloud users should consider using different wait, retry, and fallback strategies for different types of transient faults.
Resource Fault: Hardware Fault	Cloud users should consider more active backup and immediately make resources into passive/quiescent state when encountering hardware faults.
Resource Fault: Resource Control Fault	Cloud users should avoid using API calls during outage and use alternative means such as over-provisioning and zone-aware requests.

CONCLUSION

In this article, we have performed a comprehensive study on Cloud API issues and classified the failures and faults into different categories implying different remedies for cloud platforms and cloud users. Please note that the failure and fault classifications can be reapplied to different case studies. We have built solutions around a fault-tolerant version of the APIs and better exception handlings. Our general conclusion is that software engineers should treat cloud APIs as prone to various types of failures and faults and should use more defensive programming techniques or external solutions to tolerate them.

ACKNOWLEDGMENTS

This project is supported by PetroChina Innovation Foundation (Grant No. 2012D-5006-0304) and the Fundamental Research Funds for the Central Universities (Grant No. 16CX02047A).

REFERENCES

1. Q Lu et al., "Cloud API Issues: an Empirical Study and Impact," *Proceedings of 9th ACM SIGSOFT Conference on Quality of Software Architecture*, 2013.
2. A Avizienis et al., "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, 2004.
3. Z Yin et al., "An empirical study on configuration errors in commercial and open source systems," *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011.
4. X Xu et al., "POD-Diagnosis: Error Diagnosis of Sporadic Operations on Cloud Applications," *The 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014.
5. Q Lu et al., "Mechanisms and Architectures for Tail-Tolerant System Operations in Cloud," *USENIX HotCloud 2014*, 2014.

ABOUT THE AUTHORS

Zhongwei Li is an associate professor in the College of Computer and Communication Engineering at China University of Petroleum (East China). Contact him at lizhongwei@upc.edu.cn.

Qinghua Lu is an associate professor in the College of Computer and Communication Engineering at China University of Petroleum (East China). Qinghua Lu is the corresponding author. Contact her at qinghualu@upc.edu.cn.

Liming Zhu is a research scientist in Data61 at the Commonwealth Scientific and Industrial Research Organisation (CSIRO). Contact him at Liming.Zhu@data61.csiro.au.

Xiwei Xu is a research scientist in Data61 at the Commonwealth Scientific and Industrial Research Organisation (CSIRO). Contact her at xiwei.xu@data61.csiro.au.

Yue Liu is a post-graduate student in the College of Computer and Communication Engineering at China University of Petroleum (East China). Contact him at s17070790@s.upc.edu.cn.

Weishan Zhang is a professor in the College of Computer and Communication Engineering at China University of Petroleum (East China). Contact him at zhangws@upc.edu.cn.

IEEE computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.
MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.
COMPUTER SOCIETY WEBSITE: www.computer.org
OMBUDSMAN: Direct unresolved complaints to ombudsman@computer.org.
CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.
AVAILABLE INFORMATION: To check membership status, report an address change, or obtain more information on any of the following, email Customer Service at help@computer.org or call +1 714 821 8380 (international) or our toll-free number, +1 800 272 6657 (US):

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES
***Computer*:** The flagship publication of the IEEE Computer Society, *Computer*, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.
Periodicals: The society publishes 13 magazines, 19 transactions, and one letters. Refer to membership application or request information as noted above.
Conference Proceedings & Books: Conference Publishing Services publishes more than 275 titles every year.
Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.
Technical Committees: TCs provide professional interaction in more than 30 technical areas and directly influence computer engineering conferences and publications.
Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.
Certifications: The society offers two software developer credentials. For more information, visit www.computer.org/certification.

NEXT BOARD MEETING

7-8 June 2018, Phoenix, AZ, USA

EXECUTIVE COMMITTEE

President: Hironori Kasahara
President-Elect: Cecilia Metra; **Past President:** Jean-Luc Gaudiot; **First VP, Publication:** Gregory T. Byrd; **Second VP, Secretary:** Dennis J. Frailey; **VP, Member & Geographic Activities: Forrest Shull**; **VP, Professional & Educational Activities: Andy Chen**; **VP, Standards Activities:** Jon Rosdahl; **VP, Technical & Conference Activities:** Hausi Muller; **2018-2019 IEEE Division V Director: John Walz**; **2017-2018 IEEE Division VIII Director:** Dejan Milojicic; **2018 IEEE Division VIII Director-Elect:** Elizabeth L. Burd

BOARD OF GOVERNORS

Term Expiring 2018: Ann DeMarle, Sven Dietrich, Fred Douglis, Vladimir Getov, Bruce M. McMillin, Kunio Uchiyama, Stefano Zanero
Term Expiring 2019: Saurabh Bagchi, Leila DeFloriani, David S. Ebert, Jill I. Gostin, William Gropp, Sumi Helal, Avi Mendelson
Term Expiring 2020: Andy Chen, John D. Johnson, Sy-Yen Kuo, David Lomet, Dimitrios Serpanos, Forrest Shull, Hayato Yamana

EXECUTIVE STAFF

Executive Director: Angela R. Burgess
Director, Governance & Associate Executive Director: Anne Marie Kelly
Director, Finance & Accounting: Sunny Hwang
Director, Information Technology & Services: Sumit Kacker
Director, Membership Development: Eric Berkowitz
Director, Products & Services: Evan M. Butterfield

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928
Phone: +1 202 371 0101 • **Fax:** +1 202 728 9614
Email: hq.ofc@computer.org
Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 **Phone:** +1 714 821 8380
Email: help@computer.org

MEMBERSHIP & PUBLICATION ORDERS
Phone: +1 800 272 6657 • **Fax:** +1 714 821 4641 • Email: help@computer.org
Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan
Phone: +81 3 3408 3118 • **Fax:** +81 3 3408 3553
Email: tokyo.ofc@computer.org

IEEE BOARD OF DIRECTORS

President & CEO: James Jefferies
President-Elect: Jose M.F. Moura
Past President: Karen Bartleson
Secretary: William P. Walsh
Treasurer: Joseph V. Lillie
Director & President, IEEE-USA: Sandra "Candy" Robinson
Director & President, Standards Association: Forrest D. Wright
Director & VP, Educational Activities: Witold M. Kinsner
Director & VP, Membership and Geographic Activities: Martin Bastiaans
Director & VP, Publication Services and Products: Samir M. El-Ghazaly
Director & VP, Technical Activities: Susan "Kathy" Land
Director & Delegate Division V: John W. Walz
Director & Delegate Division VIII: Dejan Milojičić

CONNECT ON INTERFACE

Explore **INTERFACE**, a communication resource to help members engage, collaborate and stay current on Computer Society activities. Use **INTERFACE** to learn about member accomplishments and find out how your peers are changing the world with technology.

We spotlight our professional sections and student branch chapters, sharing their recent activities and giving leaders a window into how chapters around the globe grow, thrive and meet member expectations. Plus, **INTERFACE** will keep you informed on Computer Society–related activities so you never miss a meeting, career development opportunity or important industry update.

Connect today at interface.computer.org

IEEE COMPUTER SOCIETY
INTERFACE