IEEE CLOUD COMPUTING

VOLUME 5, NUMBER 1 JANUARY/FEBRUARY 2018

Fortifying the Cloud

IEEE IEEE computer society IEEE ComSoc IEEE Communications Society

www.computer.org/cloud

COMPSAC 2018

Tokyo, Japan

July 23-27

Staying Smarter in a Smartening World

IEEE Advancing Technology for Humanity

IEEE computer society

Call for Papers

COMPSAC is the IEEE Computer Society Signature Conference on Computers, Software and Applications. It is a major international forum for academia, industry, and government to discuss research results and advancements, emerging challenges, and future trends in computer and software technologies and applications. The theme of COMPSAC 2018 is *Staying Smarter in a Smartening World*.

Computer technologies are producing profound changes in society. Emerging developments in areas such as Deep Learning, supported by increasingly powerful and increasingly miniaturized hardware, are beginning to be deployed in architectures, systems, and applications that are redefining the relationships between humans and technology. As this happens, humans are relinquishing their roles as masters of technology to partnerships wherein autonomous, computer-driven devices become our assistants. What are the technologies enabling these changes? How far can these partnerships go? What will be our future as we deploy more and more "'things" on the Internet of Things – to create smart cities, smart vehicles, smart hospitals, smart homes, smart clothes, etc.? Will humans simply become IoT devices in these scenarios and if so, what will be the social, cultural, and economic challenges arising from these developments? What are the technical challenges to making this all happen – for example, in terms of technologies such as Big Data, Cloud, Fog, Edge Computing, mobile computing, and pervasive computing in general? What will be the role of the 'user' as the 21st Century moves along?

COMPSAC 2018 is organized as a tightly integrated union of symposia, each of which will focus on technical aspects related to the "smart" theme of the conference. The technical program will include keynote addresses, research papers, industrial case studies, fast abstracts, a doctoral symposium, poster sessions, and workshops and tutorials on emerging and important topics related to the conference theme. A highlight of the conference will be plenary and specialized panels that will address the technical challenges facing technologists who are developing and deploying these smart systems and applications. Panels will also address cultural and societal challenges for a society whose members must continue to learn to live, work, and play in the environments the technologies produce. Authors are invited to submit original, unpublished research work, as well as industrial practice reports. Simultaneous submission to other publication venues is not permitted. All submissions must adhere to IEEE Publishing Policies, and all will be vetted through the IEEE CrossCheck Portal. Papers relevant to COMPSAC topics published in IEEE journals in the last three years or accepted for publication can as well be proposed for live presentation. Further info is available at www.compsac.org.

Standing Committee Chair: Sorel Reisman, California State University, USA
Steering Committee Chair: Sheikh Iqbal Ahamed, Marquette University, USA

General Chairs: Shinichi Honiden (NII, Japan)
Roger U. Fujii, Fujii Systems, 2016 IEEE Computer Society President

Program Chairs in Chief:
Jiannong Cao (Hong Kong Polytechnic University, Hong Kong)
Stelvio Cimato (University of MIlan, Italy)
Yasuo Okabe (Kyoto University, Japan)
Sahra Sedighsarvestani (Missouri University of Science & Technology, USA)

Workshop Chairs: Kenichi Yoshida (University of Tskuba, Japan)
Ji-Jiang Yang (Tsinghua University, China)
Hong Va Leong (Hong Kong Polytechnic University, Hong Kong)
Chung Horng Lung (Carleton University, Canada)

Local Chairs: Hironori Washizaki (Waseda University, Japan)
Nobukazu Yoshioka, NII, Japan

Important Dates

Main Conference papers
Due date: 31 January 2018 (extended)
Notification: 31 March 2018

Workshop papers
Due date: 10 April 2018
Notification: 1 May 2018

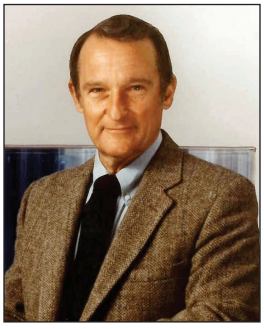
Camera Ready and Registration
Due date: May 15, 2018

Recognizing Excellence in High Performance Computing

Nominations are Solicited for the

SEYMOUR CRAY SIDNEY FERNBACH & KEN KENNEDY AWARDS

SEYMOUR CRAY COMPUTER ENGINEERING AWARD



Established in late 1997 in memory of Seymour Cray, the Seymour Cray Award is awarded to recognize innovative contributions to high performance computing systems that best exemplify the creative spirit demonstrated by Seymour Cray. The award consists of a crystal memento and honorarium of US$10,000. **This award requires 3 endorsements.**

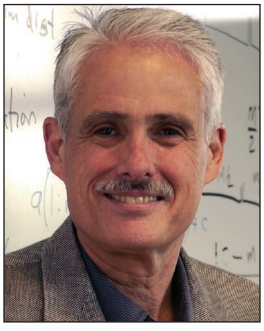
Sponsored by: IEEE computer society

SIDNEY FERNBACH MEMORIAL AWARD

Established in 1992 by the Board of Governors of the IEEE Computer Society. It honors the memory of the late Dr. Sidney Fernbach, one of the pioneers on the development and application of high performance computers for the solution of large computational problems. The award, which consists of a certificate and a US$2,000 honorarium, is presented annually to an individual for "an outstanding contribution in the application of high performance computers using innova-tive approaches." **This award requires 3 endorsements.**

Sponsored by: IEEE computer society

ACM/IEEE-CS KEN KENNEDY AWARD



Established in memory of Ken Kennedy, the founder of Rice University's nationally ranked computer science program and one of the world's foremost experts on high-performance computing. A certificate and US$5,000 honorarium are awarded jointly by the ACM and the IEEE Computer Society for outstanding contributions to programmability or productivity in high performance computing together with significant community service or mentoring contributions. **This award requires 2 endorsements.**

Cosponsored by: IEEE computer society acm Association for Computing Machinery

Deadline: 1 July 2018

All nomination details available at http://awards.computer.org

IEEE

IEEE CLOUD COMPUTING

January/February 2018
Vol. 5, No. 1

www.computer.org/cloud

TABLE OF CONTENTS

Fortifying the Cloud

Columns and Departments

Also in This Issue

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

The State of the Cloud

Mazin Yousif
T-Systems, International

In the State of the Cloud article in January 2017, I said that the cloud would become the de-facto hosting platform for all applications and social innovations. I also mentioned that it is becoming the new normal. And that does seem to be the case. Cloud computing is enabling companies to innovate at their own speed, allowing them to spin up and down images, launch applications and analytics as fast as they need. The cloud also lets consumers make choices from a wide variety of services.

Digitalization is an umbrella term that refers to all the technologies needed to become fully data-driven. There's no specific list of technologies under digitalization, but we usually include cloud, big data and analytics, IoT, intelligence, virtual reality, mixed reality, blockchain, drones, and more. Additionally, digitalization is fueling the further use of cloud computing as companies in all industry sectors are trying hard to leverage whatever data they can get hold of to offer more targeted consumer services, increase operational efficiencies, execute integration of mergers and acquisitions…, mainly to increase their bottom line and provide better experiences for their customers. Although digitalization is a long multi-phase journey with many ups and downs, it's happening everywhere around us and is expected to continuously evolve and develop.

Therefore, we expect that the adoption of cloud computing will continue because we do not see any obstacles to slow it down. In fact, adoption will accelerate further, mainly as a result of digitalization.

Let's look at the cloud computing trends we saw in 2017 and consider what we expect to see happening in 2018. I am not going to look at market shares for cloud offerings such as Amazon AWS, Microsoft Azure, and the Google Cloud Platform; rather, I will look at technologies, services and reasons for further cloud adoption.

CLOUD COMPUTING 2017

Containers were spreading like wildfire last year and clearly got elevated to mainstream adoption. Serverless computing became a standard cloud offering. Hybrid-cloud became considerably more prominent; mainly due to the release of Azure Stack, which, along with public Azure, establish a hybrid offering. We also saw private and public cloud offerings grow at a healthy pace with no indication that one would overtake the other. We saw more diversity in cloud services with additional enhancements in terms of capabilities as well as intelligence. Cloud, edge and fog were on many people's minds; sometimes causing more a great deal of confusion.

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

CLOUD COMPUTING 2018

As this year progresses, we'll see greater integration between edge and cloud – a trend that will likely increase in the coming years. Containers and serverless computing will remain integral cloud offerings. Cloud management will evolve further and incorporate intelligence and analytics to generate more insights for providers and their consumers. Cloud providers will increase the efficiency of their operations, better control energy and power consumption, and more efficiently move workloads around to better meet service level agreements (SLAs). Cloud providers will continue to tailor their services to meet consumers' needs, for example, by providing enhanced analytics or better visualization techniques for increased visibility of usage, costs, and so on. I am also hoping to see cloud brokers mature along with some pick-up in their adoption.

Cloud security will continue to evolve because it is critical to the survivability of cloud in the market and is a major barrier to the pick-up of cloud technologies. We all know that cloud datacenters get continuously attacked, but clearly cloud providers have done a good job protecting their services and consumers from such attacks. Some may have succeeded, but we have heard of relatively few major hacks in the press. The cyberattacks generated at the national level are an increasing threat as they become more sophisticated. Imagine consumers' reaction if a cloud datacenter gets hacked and all their stored data get stolen. Cloud providers must continuously evolve their cybersecurity technologies/processes and bring more intelligence and analytics to protect their datacenters. Similarly, cloud reliability and enhancements will be required to cement the availability of cloud datacenters.

Expect to see closer integration of cloud and the edge. The edge is everywhere things happen and the cloud is where the backend compute is hosted. The growth of edge computing is happening to support IoT use cases that either exhibit latencies that can't wait for roundtrip travel to the cloud or generate excessive data that doesn't need to be transferred to the cloud or for legal reasons. For example, envisioning autonomous cars becoming the indispensable future transportation method is definitely putting emphasis on further development of the edge as we imagine a future where each autonomous car becomes a micro-datacenter with enough compute infrastructures to support reliable and safe driving on the road. Another example is the use of cameras or drones in airports for security and surveillance. As the resolution of the cameras reaches 1080P or higher, it does not make sense to send all that data to the cloud. Processing can be done on the drone or a local server close to where the cameras reside. Similarly, edge will play an increasingly important role in adversarial environment such as battlefields and urban warfare.[1]

CONCLUSION

I also want to remind readers of the major themes we covered in 2017. We had special issues on Intelligence in the Cloud; Native Cloud Applications; The role of Multiclouds in the enterprise and the integration of edge and cloud computing. In 2018, we will also have four special issues with the following themes: Cloud Reliability; Convergence of IoT, Edge and Cloud Computing in Smart Cities; Reengineering Cloud Datacenters; and Biometric-as-a-Service (Cloud-based technology, systems and applications).

Starting this issue, we are presenting an additional column called "Cloud and the Community." Christine Miyachi from Xerox Corporation will serve as its editor. Her first column will discuss cloud communities of practice. We'll also continue with our regular columns, which are Cloud Economics, Cloud Standards, Cloud Tidbits; Cloud and the Law; and Cloud Research (Blue Skies). I urge all readers to look at the columns closely and share feedback on everything that we publish.

REFERENCE

1. A. Castiglione et al., "Context Aware Ubiquitous Biometrics in Edge of Military Things," *IEEE Cloud Computing*, vol. 4, no. 6, 2017, pp. 16–20.

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.

SUBMIT TODAY

IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING

SCOPE

The *IEEE Transactions on Sustainable Computing* (*T-SUSC*) is a peer-reviewed journal devoted to publishing high-quality papers that explore the different aspects of sustainable computing. The notion of sustainability is one of the core areas in computing today and can cover a wide range of problem domains and technologies ranging from software to hardware designs to application domains. Sustainability (e.g., energy efficiency, natural resources preservation, using multiple energy sources) is needed in computing devices and infrastructure and has grown to be a major limitation to usability and performance.

Contributions to *T-SUSC* must address sustainability problems in different computing and information processing environments and technologies, and at different levels of the computational process. These problems can be related to information processing, integration, utilization, aggregation, and generation. Solutions for these problems can call upon a wide range of algorithmic and computational frameworks, such as optimization, machine learning, dynamical systems, prediction and control, decision support systems, meta-heuristics, and game-theory to name a few.

T-SUSC covers pure research and applications within novel scope related to sustainable computing, such as computational devices, storage organization, data transfer, software and information processing, and efficient algorithmic information distribution/processing. Articles dealing with hardware/software implementations, new architectures, modeling and simulation, mathematical models and designs that target sustainable computing problems are encouraged.

SUBSCRIBE AND SUBMIT

For more information on paper submission, featured articles, calls for papers, and subscription links visit:

www.computer.org/tsusc

IEEE computer society

IEEE COMMUNICATIONS SOCIETY

CEDA IEEE Council on Electronic Design Automation

IEEE

Community of Practice

Christine Miyachi
Xerox Corporation

The field of Cloud Community is broad and deep. This new column will explore various communities' progress on Cloud Computing, and will compare and contrast their approaches. Look for supporting material on the IEEE Cloud Computing Website (https://cloudcomputing.ieee.org/communities-of-practice).

A colleague of mine who curates a social media site on Cloud Computing once told me: "It's getting harder and harder to find articles about cloud. People are writing about cloud-related technologies but not much about cloud computing." This supports a saying we have in the office: "Cloud computing is computing." For those of us who have been in the industry a while, it has been a quick and complete transition to using the cloud for just about everything—storage, email, virtual servers, you-name-it and you can find that it is done in the cloud. My young adult children who are software engineers don't remember a time where they didn't use cloud, for storage and for all their work and school projects. They don't even use the word "cloud." Cloud Computing is so ubiquitous, easy to use, and transparent, that it is becoming difficult to determine what is enabling all of this computing.

Behind the cloud, there are many people working, driving the ubiquity and the diversity. This column will focus on Cloud Communities of Practice and will shed light on one or more of these communities. In Alan Sill's article "Defining Our Terms,"[1] he explains that a standard is successful in part by "gathering input from multiple communities, and taking a broad-based, multi-technology approach toward implementing the identified solutions to real-world problems." Inspirational and not just for standards, this is how many difficult problems are solved. Communities build clouds and these communities consists of many groups as Figure 1 shows.

Figure 1. Various communities of practice make up the cloud computing community.

While there are many communities related to cloud computing, I will select five different communities to examine in the depth in the next five issues of *IEEE Cloud Computing* magazine.

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

EXAMPLES OF COMMUNITIES AND TOPICS

Standards

In his article on "Emerging Standards and Organizational Patterns in Cloud Computing,"[2] Sill outlines three types of standards organizations as shown in Figure 2.

- industry-based special-purpose consortia composed mostly of large-scale companies and sometimes organized around a foundation model;
- formal standards development organizations (SDOs) that exist primarily to develop standards; and
- informal community organizations that can be open to single-person developer or user input.

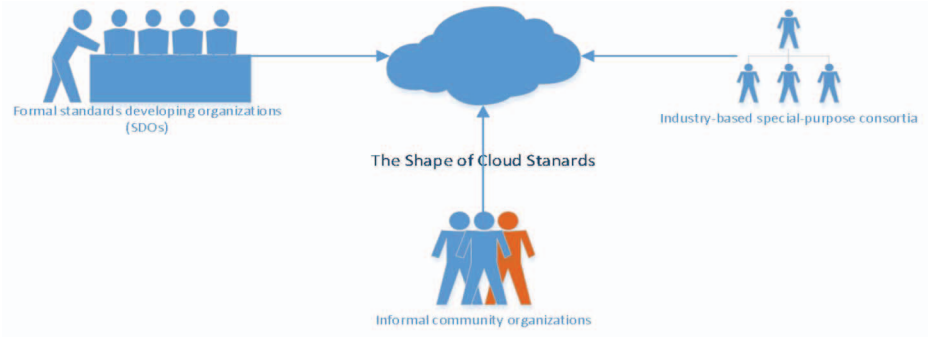


Figure 2. Three types of standards organizations.

Each of these groups is a unique community and yet they also interact synergistically to form a new community. An example of this is the *IEEE Standards Association P2302 — Standard for Intercloud Interoperability and Federation (SIIF).*[3] Members of the group range from government organizations, academia, and industry to form a group that is capable to form a strong standard that will allow clouds to work together. In addition to the series of articles in upcoming issues, I'll interview members of the team from different areas and do podcasts on the *IEEE Cloud Computing* podcast on ITunes.

Supporting Internet Technologies

Cloud computing is layered upon a long list of supporting standards. New standards emerge from different sources. This column will examine the sources that create these underlying standards, what motivates them, and how they work together. A useful list of these standards is contained in the *Inventory of Standards Relevant to Cloud Computing.*[4]

Industry Alliances and Research Groups

Many open source consortia are forming around cloud-related technologies, like The Open Fog Consortium,[5] and the Open Research Cloud,[6] which represents a "collaboration of the international community supporting scientific research computing." How do these communities form? What makes them successful? What is their output?

Cloud APIs

In the area of APIs, I will investigate a broad range of topics, such as the industry's use of Swagger[7] as within Google Cloud[8] and other cloud providers, and how RESTful APIs compare with SOAP/WSDL/XML APIs and their overall adoption.

Cloud Hardware

What does cloud hardware look like these days? As more virtualization occurs, what changes will the user experience? A colleague that provides cloud hardware farms to cloud providers laments that there is no discussion of hardware. I will look to interview cloud providers and tour data center facilities. *IEEE Cloud Computing* will be doing a special issue on Reengineering Cloud Data Centers[9] with papers due in March.

Social/Economic Communities

There are many social and economic issues related to cloud communities. I'll look at some particular areas raising concern such as monopolies or at least oligopolies in the cloud and privacy issues. How are these issues being addressed with standards? Within consortia? Through legal and regulatory actions? And what is the latest research?

Evolution of SaaS/PaaS/IaaS

This is not on the diagram, but I plan on doing a column about the foundational framework of cloud computing: SaaS, PaaS, and IaaS (Software as a Service, Platform as a Service, and Infrastructure as a Service) and related elements such as Hardware as a Service and Data as a Service. How have these essential concepts and implementations evolved over the years and where are they going? For this topic, I will interview a wide variety of cloud computing leaders and compare the different perspectives.

CONCLUSION

Thank you for reading the initial installment of my column. Please to participate in a poll I set up for readers to indicate areas that they are interested in: https://doodle.com/poll/7izfgpgtvxegqqsz.

REFERENCES

1. *Standard for Intercloud Interoperability and Federation*, standard 2302, IEEE, 2018; http://standards.ieee.org/develop/project/2302.html.
2. *Cloud Endpoints for OpenAPI Documentation | Cloud Endpoints with OpenAPI | Google Cloud Platform*, Google.
3. "Enabling Advanced IoT, 5G, AI with Fog Computing," *OpenFog Consortium*; www.openfogconsortium.org.
4. "Inventory of Standards Relevant to Cloud Computing," *StandardsInventory < CloudComputing < TWiki*, NIST; https://collaborate.nist.gov/twiki-cloud-computing/bin/view/CloudComputing/StandardsInventory.
5. "OAI/OpenAPI-Specification," GitHub; https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md.
6. *Open Research Cloud*; www.openresearchcloud.org.
7. "Reengineering Cloud Data Centers - Call for Papers," *IEEE Cloud Computing*, IEEE Computer Society, 2017; publications.computer.org/cloud-computing/2017/12/09/reengineering-cloud-data-centers-call-papers.
8. A. Sill, "Defining Our Terms," *IEEE Cloud Computing*, vol. 1, no. 1, 2014, pp. 86–89.
9. A. Sill, "Emerging Standards and Organizational Patterns in Cloud Computing," *IEEE Cloud Computing*, vol. 2, no. 4, 2015, pp. 72–76.

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.

Improving Operational Efficiency of Applications via Cloud Computing

Eric Bauer
Nokia

Editor:
Joe Weinman
joeweinman@gmail.com

Activity-based cost modeling characterizes how the levers of automation, scalable capacity, advanced self-service, agile service creation, application execution efficiency and efficiency analytics make improved operational efficiency feasible and likely for cloud-based application services.

Enterprises migrate or deploy their software-based applications to the cloud both to deliver new service and value faster to drive their top line, and to improve operational efficiency to boost their bottom line. This paper defines operational efficiency for business-to-consumer and other applications operating on public, private or hybrid cloud platforms. Simply put, it is the ratio of outputs—such as application functionality for end users, say processing claims or forecasting sales—to inputs—such as the quantity of cloud computing resources, say VMs or petabytes of storage, required to deliver that functionality. Clearly, operational efficiency can be improved by, say, eliminating wasted resources such as due to overcapacity in a private cloud.

A cost model of application service production is proposed with six efficiency improvement levers[1]: automation; scalable capacity; advanced self-service; agile service creation; application execution efficiency; and efficiency analytics. Three key indicators are also proposed to help drive continuous efficiency improvement by the enterprise: the unit cost of service production; the cost of capacity waste; and the cost of creating functionality.

THE CLOUD VALUE STREAM

Figure 1 is an input-output view of application service production on a cloud platform. The enterprise operates one or more application production chain instances which transform resource **inputs** from cloud service providers and other suppliers into application service **production output(s)**. Resource **inputs** consumed by service production accrue cost to the organization. **Production** of application capacity serves user **demand**. Production of online application capacity beyond what is required to serve actual user demand with acceptable service quality and business risk is surplus or **waste.** By analogy, **production** for an airline is flying airplane seats; resource

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

inputs include planes, pilots, fuel, but also orchestration such as flight scheduling, airline operations centers and FAA resources; **demand** is travelers willing to buy tickets, leading to seats filled with customers; and **waste** is seats flying empty. Empty seats, like unused compute resources, are particularly onerous because they represent "perishable capacity," i.e., resources that unlike, say, automobiles, can't be inventoried for future use if not needed immediately.

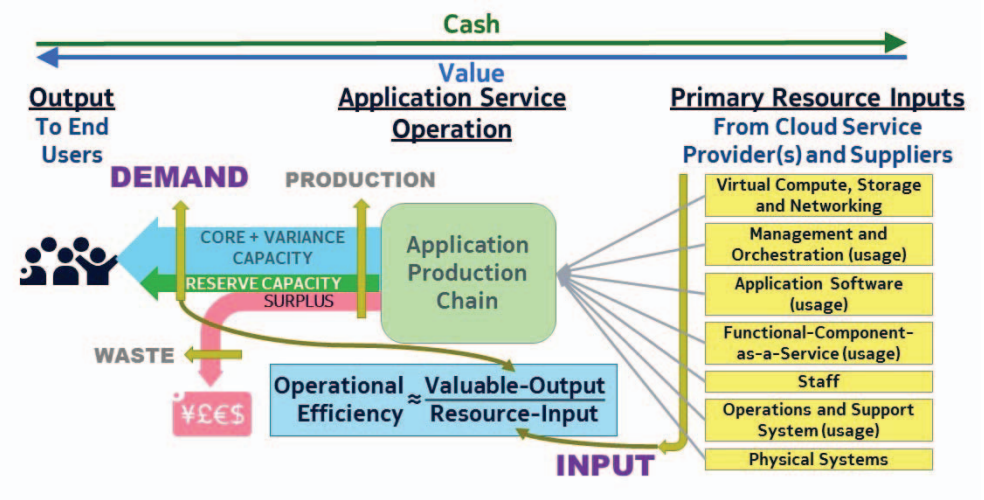


Figure 1. Operational efficiency in the cloud value stream.

Operational efficiency, as shown in Figure 1, is the ratio of valuable application output(s) serving user demand to resource input(s) consumed producing that valuable output. A simplistic strategy would be to minimize resource inputs, but the real world is more complex, requiring a balance of multiple operational, business, and customer experience criteria. Thus, a more sensible operational efficiency goal is to sustainably achieve the targeted lead time, quality, value, and customer delight at the lowest cost with acceptable business risk.

KEY EFFICIENCY INDICATORS

Organizations invest in efficiency improvement primarily to improve their financial performance and to deliver new service and value to users faster. Figure 2 simplifies the financial model for organizations offering application services: they invest to create or develop application services which are operated to serve end-user demand from which the organization derives (hopefully profitable) revenue. To continue the airline analogy, airlines invest in flight services, which are operated to serve end-user traveler demand from which airlines (hopefully) make money. To monitor and manage service-related expenses, organizations can use three key efficiency indicators, i.e., metrics: unit cost of service production; cost of capacity waste, and cost of creating functionality.

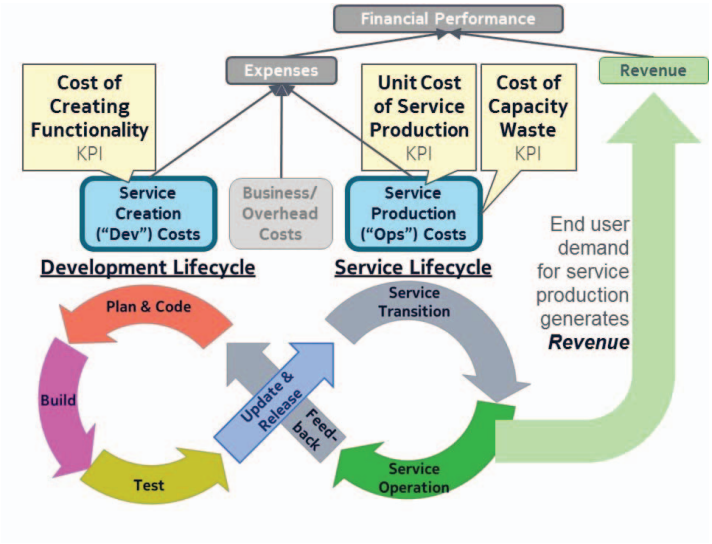


Figure 2. Key efficiency indicators of application service production.

Just as airlines improve operational efficiency by minimizing their cost-per-seat-mile while maximizing their seat occupancy level, enterprises operating applications fundamentally boost their operational efficiency by minimizing their unit cost of service production and minimizing their cost of capacity waste.

Unit Cost of Service Production

As shown in Figure 3, **unit cost of service production** is the cost of resource inputs consumed by production divided by the quantity of service output produced. This can be evaluated over a given time period, say, 1,000 hours of service delivery. Higher efficiency equates to a reduction in the intensity of resources consumed by production. In effect, these are two sides of the same coin: higher efficiency means more output for less input; lower unit cost means less input for more output. These costs come from multiple sources, such as the cost of wasted capacity and unnecessary costs in creating application functionality, and is the rationale for modern approaches such as DevOps.

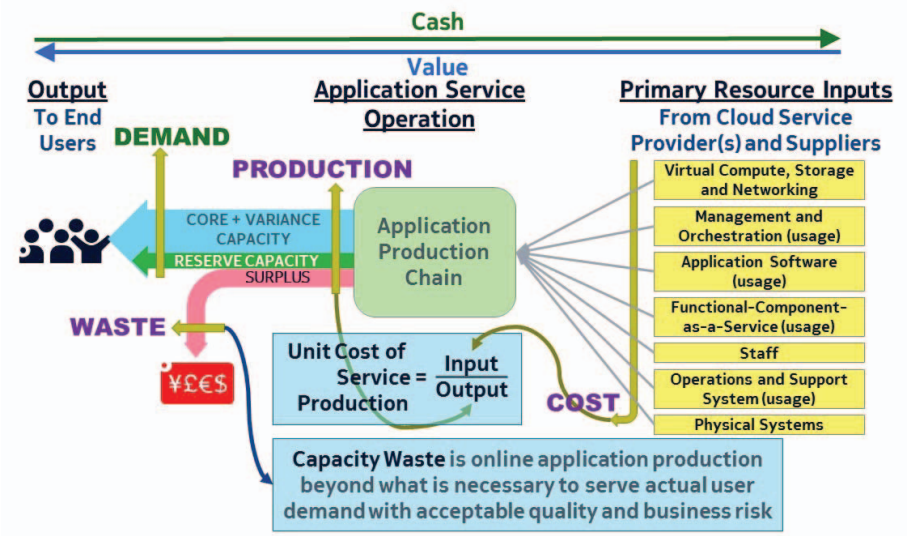


Figure 3. Unit cost of service production and capacity waste in the cloud value stream.

Cost of Capacity Waste

Rapid elasticity / scalability is the key characteristic of cloud computing[2] which enables organizations to adjust their real-time online capacity to avoid squandering cloud resources on surplus application service production. In a very analogous way, the electric power industry defines *perfect dispatch* of electricity as "the calculated, hypothetical [operation] that would result in the lowest production cost while maintaining reliability [that] could be achieved in real-time … if all system conditions, such as the load forecast, unit availability and performance, interchange and transmission outages and constraints, occurred exactly as predicted."[3] Likewise, *perfect* or *target capacity* for a cloud-based application can be defined as the capacity management plan that results in lowest production costs while delivering acceptable service quality and conforming to all service provider policies, including reserve requirements, that could have been achieved in real-time operations if all system conditions had occurred exactly as predicted.[4] As shown in Figure 3, **capacity waste** is online application capacity that is surplus to the capacity necessary to serve actual user demand with acceptable service quality and business risk, meaning greater than perfect capacity. The incremental cost to the organization of resources consumed or allocated producing this surplus online application capacity is the **cost of capacity waste.**

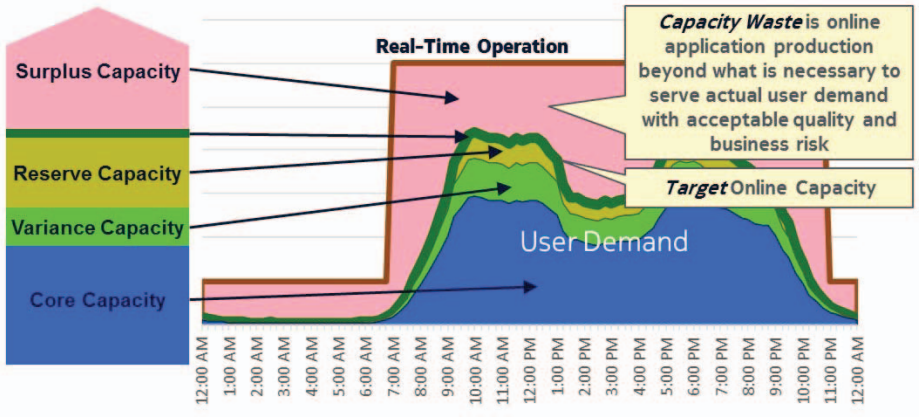


Figure 4. Example of target online application capacity and capacity waste.

Figure 4 visualizes the perfect target online application capacity[4,5] as the level of production that is necessary to serve user demand with acceptable service quality and business risk; this level of target capacity is the sum of:

- *Core Capacity* — engineered to serve average or mean user demand in a brief capacity management window (e.g., today between 9:00 a.m. and 9:15 a.m.).
- *Variance Capacity* is engineered to cover statistical peaks in user demand above average or core capacity in the capacity management window (e.g., busiest second today between 9:00 a.m. and 9:15 a.m.).
- *Reserve Capacity* is spare online capacity above core and variance capacity to mitigate the user service quality impact of:

    1. **failures**, such as redundancy for high availability like the "+K" in "N+K" load sharing arrangements of worker components;
    2. **lead time demand** — reserve capacity is engineered to cover rising demand in the interval between deciding to add additional online capacity and when that new capacity is available to serve user demand. For example, if typical lead time for capacity growth is 5 minutes, then reserve capacity should be sufficient to cover at least 10 minutes of demand growth. Faster capacity fulfillment times enable smaller levels of reserve capacity; slower capacity fulfillment times call for larger reserve capacity increments;

3. **other errors and contingencies**, such as demand surges that were not forecast and disaster scenarios.

An organization's operational policies, and overall business culture and strategic focus dictate the level of reserve capacity which appropriately balances the organization's appetite for cost savings (which drives for minimal reserve capacity) and the organization's aversion to user service quality risk (which drives for generous reserve capacity).

By analogy, the perfect tire capacity for typical automobile usage is five: exactly four tires of core capacity to carry the vehicle (i.e., zero variance capacity) and one spare tire as reserve capacity to mitigate the user impact of a flat tire. Carrying a second spare tire is wasteful because it consumes precious cargo space and fuel while offering negligible risk reduction for typical disaster scenarios.

For example, if the target online capacity to serve user demand with acceptable user quality and business risk calls for 5 load-shared virtual machine instances for some time window but real-time operation carries 20 virtual machine instances, then the incremental cost to the organization of the 15 virtual machine instances producing surplus capacity adds to cost of capacity waste. Note that the unit cost of resources consumed by wasted capacity is likely to vary across space and time; for example, a unit of compute resources wasted at 2 p.m. may cost the organization more than compute resources wasted at 2 a.m.; costs will also vary depending on cloud provider and whether they are public cloud or private cloud resources, the degree to which they benefit from a free tier, etc.

Cost of Creating Functionality

Cost of creating functionality normalizes an organization's costs to create and initially deploy new functionality to serve user or business needs, like new user features to stimulate user demand or automation to reduce unit cost of service production and minimize cost of capacity waste. It costs more efficient development teams less money and time to create a given amount of functionality with acceptable quality.

COST MODEL OF APPLICATION SERVICE PRODUCTION

A cost model characterizes the transformation of resource inputs into valuable outputs to enable methodical measurement, management and improvement.[6] Figure 5 visualizes the dynamic, time-driven, activity-based cost model for production of cloud-based application services as a four-layer cost assignment network.[2]

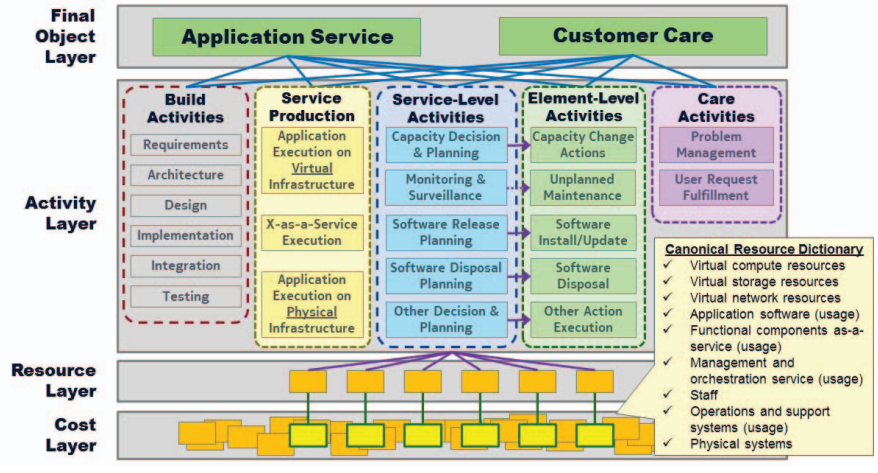


Figure 5. Dynamic, time-driven, activity-based cost model of application service production.

- **Final object layer** – represents the fundamental production outputs from a digital service provider:

  1. *application service* – the output of the application production chain.
  2. *customer care* to assure user satisfaction and support technical demands of users.

- **Activity layer** – application service and customer care final objects result from the following service creation and production activities:

  1. *Build activities* for requirements, architecture, design, implementation, integration and testing.
  2. *Service production* for pure execution of application software components executing on either cloud infrastructure or traditional, physical hardware platforms. This category also considers functional components offered as-a-service (a.k.a., PaaS) by some cloud service provider.
  3. *Service-level activities* for information analysis and strategic decision and action selection of service operation activities like monitoring and surveillance of application service components.
  4. *Element-level activities* for tactical action coordination, execution and oversight of operational activities on application or component instances, like elastically scaling an application component instance or repairing a failed component instance.
  5. *Customer care activities* to assure user satisfaction and support technical demands of users.

- **Resource layer** – abstractly characterizes the resources that are directly used or consumed by the activity layer:

  1. Virtual compute resources.
  2. Virtual storage resources.
  3. Virtual networking resources.
  4. Application software usage charges.
  5. Functional components consumed as-a-service (a.k.a., PaaS), such as database-as-a-service or load-balancing-as-a-service.
  6. Management and orchestration service which supports automated operations.
  7. Human staff.
  8. Operations and support systems, such as trouble ticketing systems.
  9. Physical systems.

- **Cost layer** – activities use or consume actual, rather than abstract, resources that are typically offered by cloud service providers or other suppliers. The cost layer models the compensation that the organization offers to the suppliers and providers of consumed resources. Costs for actual resources often vary between suppliers and may vary based on time, location, intensity of consumption, pricing plan and other factors. The organization's supply chain management team will often select qualified suppliers and pricing plans that minimize the organization's overall expenses.

EFFICIENCY IMPROVEMENT LEVERS

Having characterized the transformation of costs and resource inputs into valuable application service production output via the dynamic, time-driven, activity-based cost model, we see in Figure 6 that organizations can improve the efficiency of service creation and operation activities via six levers:

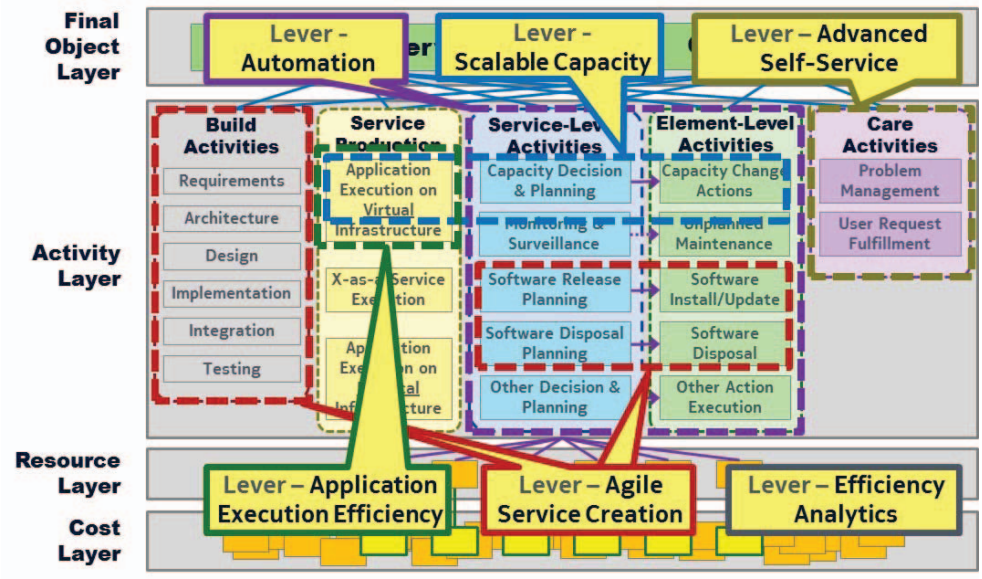


Figure 6. Efficiency improvement levers for application service production.

- **Automation** – automating data collection, analysis, strategic decision and action selection and tactical action coordination execution and oversight of service-level and element-level activities improves efficiency via:

  1. Reducing intensity of labor input for service operations.
  2. Improving right-first-time execution quality to reduce costs of reworking faulty automated actions.
  3. Reducing non-productive resource consumption for work-in-process (WIP) via faster execution.

- **Agile Service Creation** – improving effectiveness of developing and deploying new services and value, as well as efficiency via:

  1. Reducing intensity of labor input for service development, deployment, checkout & disposal.
  2. Improving right-first-time quality of service development, deployment, checkout & disposal to reduce costs of poor quality.
  3. Reducing time non-productive resources that are held as work-in-process during service development, deployment, checkout & disposal.

- **Scalable Capacity** – elastic scalable capacity management enables real-time online application service capacity to closely track perfect target online capacity to minimize the organization's cost of capacity waste.
- **Application Execution Efficiency** – efficiency of application service production is improved by selecting and deploying application elements featuring both efficient steady-state operation and favorable dynamic operational characteristics. For example, better right-first-time execution quality of an element's lifecycle management actions like scaling reduces the frequency of costly and time-consuming rework actions.
- **Advanced Self-Service** for user actions can both reduce intensity of organization's labor input for service encounters and can increase end users' value-in-use via co-production of improved service experiences.
- **Efficiency Analytics** facilitate objective and quantitative measurement, analysis, prediction and improvement of service operations costs to drive continuous efficiency improvement.

CONCLUSION

Operational efficiency of application service production is thus improved via:

- **Reducing unit costs** of service production and service creation, and
- **Reducing waste**, especially: overproduction of application service output; squandering or inefficiently using resource inputs; costs of poor quality; excess time consumed by activities; and creation of functionality that is not valued.

Measuring and managing unit cost of service production, cost of capacity waste and cost of creating functionality enables organizations to methodically improve their operational efficiency. See *Economic Efficiency of Cloud-Based Application Services*[1] for further information.

REFERENCES

1. E. Bauer, *Economic Efficiency of Cloud-Based Application Services*, 2017; doi.org/dx.doi.org/10.13140/RG.2.2.22188.56969.
2. *Cloud computing -- Overview and vocabulary*, ITU-T standard ISO/IEC 17788, International Organization for Standardization & International Electrotechnical Committee, 2014.
3. *Perfect Dispatch Factsheet*, PJM, 2014; https://learn.pjm.com/-/media/about-pjm/newsroom/fact-sheets/perfect-dispatch-fact-sheet.ashx.
4. E. Bauer, *Lean Computing for the Cloud*, Wiley-IEEE Press, 2016.
5. *International Good Practice Guidance: Evaluating and Improving Costing in Organizations*, Professional Accountants in Business Committee, 2009.

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. He received his BS in Electrical Engineering from Cornell University and his MS in Electrical Engineering from Purdue University. Bauer has been awarded 24 US patents. Contact him at Eric.Bauer@nokia.com.

Holistic Workload Scaling: A New Approach to Compute Acceleration in the Cloud

Juan F. Pérez
Universidad del Rosario
Bogotá, Colombia

Lydia Y. Chen
IBM Research Zurich
Rüschlikon, Switzerland

Massimo Villari
University of Messina, Italy

Rajiv Ranjan
Newcastle University, UK

Editor:
Rajiv Ranjan
rranjans@gmail.com

Workload scaling is an approach to accelerating computation and thus improving response times by replicating the exact same request multiple times and processing it in parallel on multiple nodes and accepting the result from the first node to finish. This is not unlike a TV game show, where the same question is given to multiple contestants and the (correct) answer is accepted from the first to respond. This is different than traditional strategies for parallelization as used in, say, MapReduce workloads, where each node runs a subset of the overall workload. There are a variety of strategies that trade off metrics such as cost, utilization, performance, and interprocessor communication requirements. Performance modeling can help determine optimal approaches for different environments and goals. This is important, because poor performance can lead to application and domain-specific losses, such as e-commerce conversions and sales.[1] Performance modeling and analysis plays an important role in designing and driving the selection of resource scaling mechanisms. Such modeling and analysis is complex due to time-varying workload arrival rates and request sizes, and even more complex in cloud environments due to the additional stochastic variation caused by performance interference due to resource sharing across co-located tenants. Moreover, little is known on how to multi-scale, i.e., dynamically and simultaneously

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

scale resources vertically, horizontally, and through workload scaling. In this article, we first demonstrate the effectiveness of multi-scaling in reducing latency, and then discuss the performance modeling challenges, particularly for workload scaling.

A study from Amazon estimates[1] a latency delay of 100 ms can cause a one percent sales drop. A recent study from Akamai[1] shows that a one second delay in page response can result in 7% loss in e-commerce conversions. These numbers illustrate the relevance of finding novel solutions to the long standing and critical challenge of reducing and/or guaranteeing the latency of interactive applications, such as web services, in a cost-effective way. Traditionally, the fundamental difficulty lies in the workload volatility, i.e., time-varying request arrival rates and varying request sizes being served by resources that may already be partially loaded. Cloud computing, with its unique ability to scale resources on demand, offers a powerful engineering solution to tackle the workload variability, but possibly by incurring additional costs due to pay-per-use pricing. The advancement of virtualization technologies makes a wide range of resources readily available upon users' requests, e.g., virtual machines, CPU cores, and docker images, and "serverless compute APIs" which can be triggered to react to changes in the workload.

However, the downside of the cloud is that the performance of virtual resources may not be stable[2] due to the underlying hardware, colocated applications, virtualization solutions, and network congestion spikes. For example, the performance of web services can be significantly degraded by CPU or network hungry neighbor VMs due to the resource contention.[3] Such issues become even more prominent when moving into multitenant clouds, where the degree of resource sharing increases significantly. The impact of this problem is more tangible for the tail latency, e.g., 95th or 99th percentile, which can grow much larger than the average latency, hindering the users' quality of experience significantly. The pitfall of resource sharing presents itself as a challenge to manage and model interactive applications[4,5] as the service rate per virtual resource is no longer constant making the service times become even more volatile.

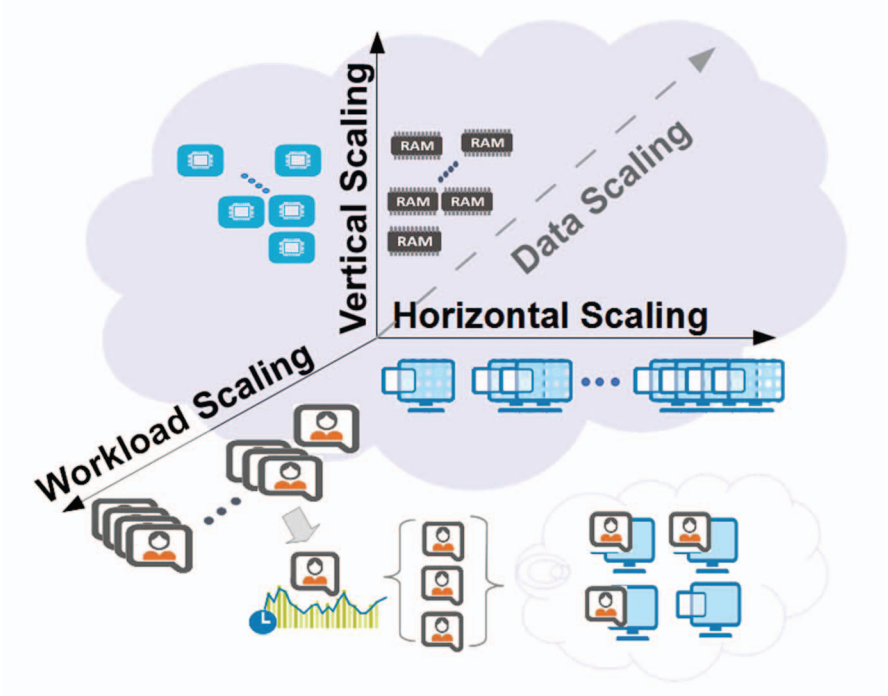


Figure 1. Scaling choices for interactive applications on the cloud.

To defend the latency against the workload variability, particularly the arrival pattern, a plethora of prior art[2,3] develops auto-scaling solutions along the vertical and horizontal directions. Vertical scaling increases/decreases virtual resources per virtual machine, e.g., the number of virtual cores, whereas horizontal scaling increases/decreases the number of virtual machines. The core principle behind these solutions is to scale the resources according to the workload demands and the latency target, often with a focus on the average value. While resource scaling can cost-effectively fulfill the average latency target for interactive services, it falls short in curtailing the *tail* latency due to the high variability per virtual resource.

Consequently, workload scaling has emerged as an alternative to manage the tail performance at scale,[6] specifically targeting scenarios where servers have time-varying processing speeds as in the cloud. Quite differently from resource scaling, workload scaling deliberately increases the workload by cloning incoming requests and simultaneously processing them on different virtual or physical servers. A reply can thus be sent to the user as soon as the *first* clone completes execution, that is, the fastest clone determines the request processing time. The *advantage* of workload scaling thus lies in making the most of the available resources by executing the same request on several servers and using the fastest to respond. The *drawback* of workload scaling is the additional load introduced by the clones, requiring underutilized servers or elastic resources to be available. This solution thus requires careful use as it could potentially *harm* the application performance if used in peak-load conditions.

In this article, we first explain the advantages and limitations of scaling resources and workloads through empirical examples of web services. We then discuss key modeling challenges found when capturing vertical, horizontal, and workload scaling, with a focus on the latter as it is the least studied.

THE LANDSCAPE OF SCALING SOLUTIONS

In this section we show the limits of vertical and horizontal scaling and how these can be combined with workload scaling to robustly manage the tail latency. We focus on the latency mean and 95th percentile as key performance metrics, and employ web applications hosted in the cloud, modifying the number of cores per VM (vertical scaling) and the number of VMs (horizontal scaling). We close this section by highlighting the limitations of these solutions.

Vertical Resource Scaling. In Figure 2, we show the latency 95th percentile observed for RUBiS, a web shopping benchmark, on a single VM at a private cloud where a neighboring VM workload is injected to emulate inference. Requests are generated at a constant arrival rate, i.e., 100 requests per second, while the number of virtual cores increases from two to six cores, one at a time every 6 minutes. One can clearly see that the tail latency decreases with the increasing number of virtual cores, but with a decaying marginal gain. This is because the processing time, i.e., latency, follows a $1/n$ rule where n is the number of cores. It will be apparent that the latency improvement between 5 and 6 virtual cores is quite small, compared to the difference between 2 and 3 cores. The effective capacity per VM is not linearly proportional to the number of virtual cores per VM, even though the price does increase linearly with the number of cores. In fact, even the largest relative gain in latency (obtained when moving from 2 to 3 cores) is not proportional to the increase in cost. Thus, increasing the number of cores may have a limited impact and may not be cost effective. Further, increasing the number of cores does not offer a solution to the larger latency due to the interference caused by the neighboring VMs.

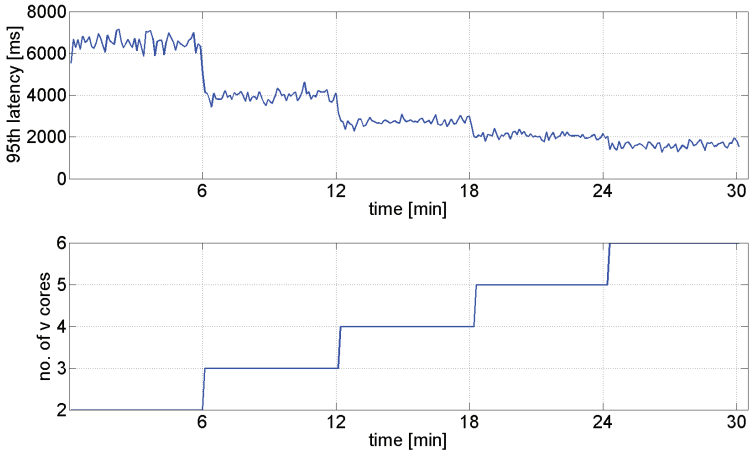


Figure 2. The tail latency of RUBiS request when increasing the number of allocated virtual cores, under a constant arrival rate of 100 requests per second.

Horizontal Resource Scaling. In Figure 3, we show the improvement in the average latency of a popular web knowledge application, MediaWiki,[7] as the number of VMs increases. Two types of VMs are considered, namely wimpy and brawny, which consist of 2 and 4 virtual cores, with 2 and 4 GB RAM, respectively, to serve a load of 10 requests per second. We observe a similar trend as for vertical scaling: the latency decreases with the increasing number of VMs but the marginal gain decreases even as the cost increases. The best latency provisioning wimpy VMs (5) is around 138 ms, whereas provisioning 5 brawny instances results in an average latency as low as 120 seconds. Moreover, brawny VMs achieve lower latency than wimpy VMs for any given number of instances. However, the improvement is definitely less than half even though the number of virtual resources doubles and the prices also doubles according to the standard market practice, e.g., Amazon EC2. One may thus conclude that using a sufficient number of brawny instances, i.e., 5, one can achieve the target latency of 120 ms, whereas even a high number of wimpy instances fails to achieve such a target. If we combine horizontal scaling of wimpy VMs with a workload scaling factor of two a surprising result occurs. Upon arrival of requests, we replicate them once and send each replicas to two different VMs. The latency is determined by the fastest request among the two replicas. One can see that a sufficiently large number of wimpy instances, i.e., 5 VMs, can achieve an average latency as low as 115 ms, which is lower than the target and better than the best performance achieved with brawny instances.

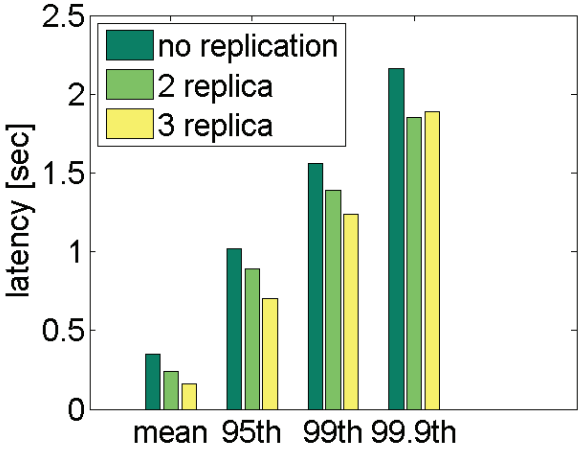


Figure 3. The average latency of hosting MediaWiki in cloud under three scaling strategies: (i) adjusting wimpy VMs only, (ii) adjusting brawny VMs only, and (iii) adjusting wimpy VMs with a replication factor of two.

At a first order of analysis, the latency of the system can be modeled by the theory of order statistics, which describes among other things the expected value of the minimum of k samples taken from a given distribution. For example, the expected value of the minimum of k samples taken from a uniform distribution on [0,1] is $1/(k+1)$. In the real world, such a simple model is insufficient and performance modeling is required, because we also note that when the number of wimpy instances is low, 2 instances or less, replicating queries results in a latency worse than without request replication. This can be explained by the extra load introduced by the

replicated requests, and indicates that enabling workload scaling without sufficient resources can result in more harm than performance advantages.

Workload Scaling. As shown in the previous example, and in agreement with prior art, speculatively replicating requests is an effective strategy to strengthen system dependability[8] and to improve the latency,[9] particularly its high percentiles. Workload scaling policies in interactive systems can be grossly classified by the issuing time of the replicated requests and by the canceling policy on the remaining redundant requests. Replicated requests can be issued proactively upon the arrival of requests or reactively after observing performance degradation so as to minimize the processing overhead. Upon receiving the first result from replicated requests/jobs, the majority of replication policies leave the rest of replicas in the system due to the overhead of terminating requests, while a few studies show the benefits of terminating requests for certain benchmarks.[6] Additional details on these policies are provided in the section on Research Challenges and Open Issues.

Table 1. Comparisons of the analytical models on replication.

Article	Assumption		Replication		Res Scaling		Testbed
	Arr.	Proc.	Cancel	Latency	Verti.	Horiz.	Validation
[9]	Static	Ind.	√	mean			√
[4]	Static	Ind.		mean			
[10]	Static	Corr	√	mean			
[11]	Static	Ind.		dis.		√	
[5]	Dynamic	Ind.	√	mean		√	√

Optimal Workload Scaling. Though the prior art demonstrates the effectiveness of workload scaling, little is known on determining the optimal level of scaling or replication. The overhead of a high replication level can overturn the benefit of returning the first-completed request. Moreover, it also depends on the metrics of interests, i.e., the optimal level for mean latency may not be the same for the 95th or 99th percentiles. Herein, we show the performance of a MediaWiki cluster hosted on Amazon EC2, subject to different levels of workload scaling. This cluster has seven t1.micro instances: six running the MediaWiki stack and one used as central queue to dispatch the requests. Figure 4 depicts the mean and tail latency with different percentiles when applying between 1 and 3 replicas for an arrival rate of 1.33 requests per second. Clearly, replication is effective in reducing the latency tail, with reductions close to 15% with 2 replicas. However, introducing a third replica hurts the tail even if it improves the mean latency. This highlights the importance of developing analytic models that are able to compute key latency metrics, both mean and distribution, under workload scaling.

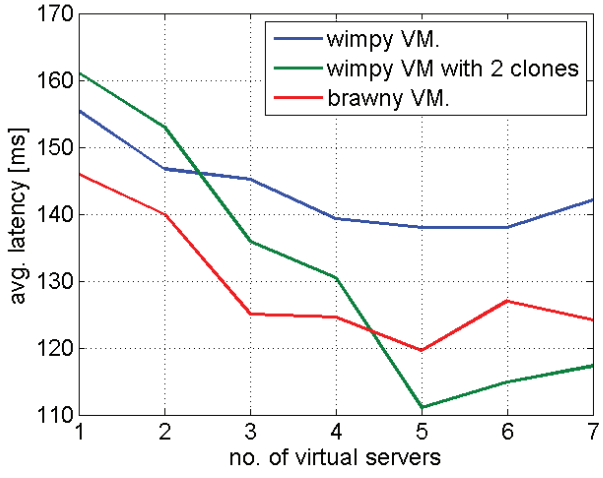


Figure 4. Different latency metrics under workload scaling only: no replication, two replicas and three replicas.

Principle and Limitation of Multi-Scaling. The main idea behind workload scaling is to increase the likelihood that requests are processed on the fastest servers. This could be determined *a priori* by a scheduling and load balancing strategy that sniffs the speed of servers via different performance indicators. In contrast, workload scaling defeats the variability by trying out multiple servers simultaneously, determining the fastest server *a posteriori*. A common criticism is therefore the high processing overhead introduced by redundant requests. In fact, the prerequisites for workload scaling to be an effective solution are (i) high variability of server speeds across VMs over time, (ii) somewhat low system load, and (iii) a sufficient number of servers available on a cost-effective basis. In the case where all the request clones are processed in full, the total load introduced by proactive replication is simply the baseline load multiplied by the replication factor. Being able to cancel the request clones (after the first one finishes) and reactively spawning the replicas can reduce the overhead. The number of available servers not only constraints the maximum number of workload scaling levels, i.e., the number of replicas cannot exceed the number of available servers, but also implicitly requires data availability. For example, when deploying the MediaWiki stack on multiple VMs, one shall ensure the all database entries are replicated on each VM, otherwise requests can only be served by a subset of VMs that contain the requested data. Hence, in addition to workload and resource scaling, data content scaling and data access network scaling are additional considerations yet to be addressed the multi-scaling in the cloud. Further details on this and other key aspects in modeling workload and multi-scaling are treated in the next section.

RESEARCH CHALLENGES AND OPEN ISSUES

In this section we delve into the challenges of modeling multi-scaling for software applications in the cloud, existing solutions and open problems. Rather than considering problems that are common to the modeling of software applications in general we focus on those key features that are more prominent when modeling multiscaling solutions. We provide a summary of these key features in related work in Table I. In the following we detail those features from the perspective of workload scaling that implicitly requires resource scaling.

Request Canceling Policy

When introducing workload scaling, multiple clones of a request are issued and processed. Since it is sufficient to receive a single response from any of these clones, it is in principle possible (and desirable) to kill or cancel all remaining clones when the first one completes service. Canceling is particularly appealing as it limits at least a portion of the additional load introduced by the workload scaling mechanism. As a result, most modeling works,[4,10] assume that request canceling is adopted, and in many cases the cancellation is assumed to have zero cost.

Canceling is also appealing from a modeling point of view and it is central in the few analytical results available for workload scaling. For instance, canceling allows Joshi and colleagues[10] to reduce a multi-server setup to a single-server one and to provide approximate solutions. Also, the solution to the Markov chain model proposed by Gardner et.al.[4] relies on the canceling assumption. One key reason why canceling simplifies the analysis is that all clones of a request finish at the same time, such that the minimum service time among all clones of a request determines the service and departure times of *all* clones.

However, canceling request clones is not always feasible as it requires incorporating into the application a functionality that is not necessarily available. Also, in "fast" distributed applications, where the request processing times are very short, canceling requests may be infeasible as the signaling delays may be too long for the signals to arrive before the processing completes. From a modeling perspective there may be stochastic variation in the delays for a cancellation signal to arrive across variably congested or distant network links to different nodes. Thus, in many applications one needs to account for the possibility that clones cannot be canceled or that the cancellation has a non-negligible cost.

Leaving all clones to execute until each of them if fully processed means that the system remains busy a much larger fraction of time than when canceling is in place. In other words, if each request is cloned (on average) *r* times the offered load increases r times. Thus, workload scaling without canceling will often require more resources than with canceling, sometimes many more, and its performance advantage is therefore limited to systems facing a low-to-moderate load, as has been shown for instance in Vulimiri et.al.[9], Qiu et.al.[8], Pérez et.al.[5].

The operation without clone canceling has also proven more difficult for modeling and analysis. In fact, the analytical results available are limited to scenarios with fairly strict assumption, such as the 2-server system with a centralized queue, Poisson arrivals, and exponential processing times considered in Lee et.al.[12]. An approximate analysis method proposed in Vulimiri et.al.[9] consists of assuming that all servers operate independently, receiving a fraction of the total traffic augmented by the workload scaling factor *r*, computing the latency for each server and obtaining the latency of a request as the minimum of the latency of *r* independent random variables, each holding the latency of a single server. The proposal in Vulimiri et.al.[9] exploited the fact that the latency offered by a single server with exponentially-distributed processing times and Poisson arrivals is itself exponentially distributed, whereas in Qiu et al.[13] a similar argument is used for phase-type distributed processing times.

Load Balancer Scaling Awareness

In a distributed setup the load balancer plays the key role of forwarding the incoming requests to the available servers. When horizontal scaling is introduced, the load balancer needs to stay aware of the changing set of resources, and appropriately modify its routing policy to incorporate and remove target resources in real time. In addition, if workload scaling is introduced, the load balancer can be further involved in properly allocating the clones of a request to the target resources. For instance, it is desirable if not mandatory that clones of the same request are processed by *different physical* servers to exploit the benefits of workload scaling, as otherwise copies of the same request would simply contend for resources at the same server.

As a result, from a modeling standpoint, the awareness of the load balancer needs to be considered explicitly as this vastly affects the performance of the scaling mechanism[14]. In the case of a scaling-unaware load balancer, a distributed server may receive multiple copies of the same request, which impacts its arrival stream creating batch arrivals and therefore impacts the latency negatively. Instead, this scenario is not possible in a scaling-aware load balancer, where arrivals to a server occurs as singletons and with the same timestamps as the request arrivals to the load balancer.

One further consideration in a distributed setup is that, in case all servers are busy, the incoming requests could either queue at the load balancer or immediately dispatched to an appropriately chosen server (e.g., following the least-connections rule). This decision typically depends on the type of application and the ratio between the transfer delays and the request processing times. If transfer times are significant, immediately dispatching the request is preferred to avoid additional delays. In either case, this choice must be capture by the model, especially when workload scaling is introduced. With the queueing-at-the-load-balancer option and workload scaling, it is possible for the load balancer to dispatch the clones of a request to the next available server as long as the reply for that request has not been received. In other words, if a request is quickly processed and copies of that request are still queueing at the load balancer, it is possible to remove those copies from the queue and avoid unnecessary processing. This is not possible if the load balancer immediately dispatches all copies to the target servers. Whereas exact numerical models exist for the former case[11], the latter option has proven more difficult due to the multi-dimensionality of the problem (many servers each with its own queue) and the close connection that exists among them since the request latency is given by the clone that finished first, in any of the servers.

The final issue to consider, mainly due to evolution of container-based microservices application architectures, is that the underlying load-balancer can either operate at transport layer (L4, request agonistic) or application layer (L7, request aware). In the case of transport layer (L4) load balancer, the incoming requests are distributed across web and/or app servers without necessarily analyzing the composition of the request type. On the other hand, the application layer (L7) load

balancer balances the requests across web and/or app servers more intelligently after analyzing the HTTP and/or HTTPS request headers. In the case of a scaling servers connected to the request agnostic (L4) load balancer, the load balancer only needs to be aware of the contact end-point of new servers as these have exact replicas of the data-sets and files. Instead, this scenario is not possible in a request aware (L7) load balancer, as the servers manage different data-sets and files (e.g., an image server vs. an account server). As a result of this, scaling in the context of L7 the load balancer will require both the contact end-point of the new servers as well as the type of request that they can serve, to be configured within the load-balancer at run-time. In summary, such diversities in load-balancing approaches further complicate the scaling and replication of web services in the cloud for improving end-to-end request latency.

Request Processing Times

As in general software application modeling, appropriately capturing the request processing times has a large impact in the accuracy of the latency predictions obtained by the model. The more general models are however more limited in the analyses that can be performed, in many cases requiring approximated methods. Thus, for multi-scaling, as in general software application modeling, many more results are available for simpler assumptions on the processing times (exponentially distributed) than for more general assumptions.

Beyond the distribution of the processing times, one key issue in modeling workload scaling is recognizing that clones of the same request may have similar processing times. For instance, if the processing time is dominated by the search time of data items and each request has a list of data items to retrieve, it is natural that all its copies have similar data retrieval and therefore overall processing times. This behavior can be captured by representing the processing times of the clones of a request as *correlated* random variables, in contrast to the usual assumption of independent processing times.

Introducing this correlation is however difficult as standard queueing models used to represent software applications assume independent request processing times. Even those models that assume correlated processing times typically assume a general correlation structure for all requests processed by the system. Instead, in the case of workload scaling the processing times of the clones of the same request can be heavily correlated whereas the processing times of different requests may still be fairly independent. The majority of modeling work still assumes independent service times, except for Qiu.et.al.[8].

Application Topology and tier-specific workload multi-scaling

The vast majority of existing works in workload scaling focuses on relatively simple applications where the whole functionality is contained in a single box. As a number of these boxes is available, a request clone can be forwarded to and processed by any of these boxes, simplifying the modeling and analysis. However, many applications do not fall within these simple assumptions. This and the following subsection consider two common cases where it is not possible to assume that a request clone can be processed by any application server.

Here we focus on the case where the application servers are actually split among groups (e.g. web tier, app tier, database tier) that provide different functionality, as in the common multi-tier architecture. In this setup a number of new questions arise regarding to workload scaling and modelling. For instance, if a request is replicated at the first tier (the one first hit by any incoming request) its clones generate additional processing requirements at all the application tiers. Further, clones could be generated at any tier and their impact therefore affects all tiers downstream from the one that implements the cloning, and the number of clones grows as the product of the replication factor applied at each tier. For instance, in a 3-tier application, if tier 2 uses cloning factor 2 (creates 1 additional clone) and tier 3 uses cloning factor 3 (creates 2 additional clones), tier 1 will not see any additional load, tier 2 will see its load doubled, while tier 3 will see its load multiplied by a factor of 6 since each of the 2 clones in tier 2 generates requests to tier 3, each of which is cloned 3 times. Moreover, as each application tier has different workload

behavior and resource requirements, it is important to develop tier specific workload scaling models.

To the best of our knowledge, the only work that explicitly considers workload scaling and modelling for a multi-tier application is sPARE[14], which proposes an algorithm to determine the best cloning levels at each tier explicitly considering the multiplicative impact of cloning at multiple tiers.

Performance Characterization

The vast majority of the existing work tends to focus on modelling the performance (e.g, request processing latency, memory utilization, CPU utilization, request processing throughput) of multi-tier applications (as noted above) on cloud datacenters that provide hypervisor-based virtualization technologies. Hypervisors enable cloud providers to create unique virtual machines (VMs) that share a set of physical hardware resources (CPU, memory, network, and disk). The hypervisor-based virtualization has many advantages (such as stronger security context), but at the cost of performance overhead, as each VM has its own Operating System (OS) image. To narrow the gap between performance of applications on virtualized and non-virtualized physical resources, container-based (e.g., Docker, LXC) cloud virtualization solutions have evolved in the last 5 years. Although containers are becoming an attractive choice for deploying applications in the cloud, very limited modelling techniques exist[15] in the literature that can appropriately distinguish and reason about the differences in the performance overhead between hypervisor-based and container-based infrastructures. For instance, hypervisors provide each VM with its own resources, while Container engine (equivalent of hypervisor) share a single host's resources among multiple containers (roughly equivalent to VMs) via *cgroup* and *namespace* mechanisms. Considering another example, the run-time performance (e.g. request processing latency) of a containerized web or database microservice not only depends on the resource configurations (e.g., CPU Cores, memory size) allocated to its *cgroup* but also on whether the underlying physical host is hypervisor-based or is hypervisor-free. In the case where containerized web services are deployed on hypervisor-based cloud resources, one would need to undertake following multi-level modelling across each application-tier including: (i) container-level and (ii) VM-level.

Data Availability

As described in the previous subsection, when modeling workload scaling it is common to assume that a replica clone can be forwarded to any of the application servers. Even if the model explicitly considers the application tiered architecture, or if the application has a single tier, a specific request may only be processed by a subset of all servers. This is the case when the application considered, or one of its tiers, serves data items stored across a number of distributed nodes such that (each of) the items required to fulfill a request are available in just a few of the nodes.

Incorporating data availability in modeling workload scaling introduces too the issue of data popularity, as this makes the request arrival rates to be much larger for popular data items than for non-popular ones. Thus, whereas introducing cloning for unpopular data items may have little to no effect, cloning requests for popular data items must consider the availability of enough servers holding that item to fully exploit the benefits of workload scaling as well as to avoid potential server overloads. As of now, this topic has not been explicitly considered in the literature on modeling workload scaling.

Reliability

Most analyses of workload scaling have focused on the performance benefits of this strategy, as it has proven effective in reducing latency. However, workload scaling has the potential of improving reliability in failure. Consider for instance a request that is cloned and its two copies are sent to two different servers. If one of the servers fails, the request clone sent to the other server can still complete processing and the reply can be sent back to the user without requiring any additional steps. Or, for error-prone systems, a voting mechanism can determine the correct result

from, say, a majority of the correct and incorrect results received. Workload scaling is thus able to boost both performance and reliability. Evaluating workload scaling across both of these perspectives simultaneously still remains largely unexplored, with few exceptions such as Qiu and Pérez[16].

Performance Metrics

As workload scaling is mainly put forward as a method of reducing response time at the possible expense of higher resource requirements, models are typically geared towards obtaining latency metrics. In many cases the metric of interest is the average latency[9,4,5]. However, workload scaling can be particularly effective to limit the tail latency, which makes obtaining the latency distribution (or its high percentiles) necessary to assess this potential benefit. Several works[11,5] have therefore focused on deriving models to obtain the latency distribution. Finally, one metric that has proven very useful[9,8] to assess when to activate a workload scaling mechanism is the *threshold load*, that is, the maximum load at which introducing one additional clone remains beneficial without congesting or overloading the total system. Obtaining this metric allows for the design of adaptive controllers that increase/decrease the cloning intensity according to the observed load, such that cloning is deactivated under peak loads to avoid overloads whereas time periods with low loads (valleys) can benefit from heavy cloning.

CONCLUSION

As we have seen, a number of recent works consider workload scaling a mechanism to latency management. A few of them further attempt to answer the fundamental question of the optimal number of replicas under various system assumptions (request canceling policy, load-balance awareness, etc.). Although the number of servers is a usual input to the proposed models, almost no work explores the resource and workload scaling jointly, except Qiu and Pérez[11], Pérez et.al.[5] . Among the latter, Qiu and Pérez[11] consider a stationary workload and is validated via simulations only. Instead, DuoScale[5] considers both horizontal resource and workload scaling and its results are experimentally validated on a testbed subject to a dynamic workload. To the best of our knowledge, there is no modeling work jointly exploring the three dimensions of horizontal resource scaling, vertical resource scaling, and workload scaling.

ACKNOWLEDGEMENTS

The research presented in this paper has been supported by the Swiss National Science Foundation National Research Programme “Big Data” (NRP 75) (project 407540 167266).

REFERENCES

1. *K. Metrics*, blog; https://blog.kissmetrics.com/loading-time.
2. M. Björkqvist, L.Y. Chen, and W. Binder, “Opportunistic Service Provisioning in the Cloud,” *Proceedings of the IEEE 5th International Conference on Cloud Computing* (CLOUD 12), 2012, pp. 237–244.
3. R. Nathuji, A. Kansal, and A. Ghaffarkhah, “Q-clouds: managing performance interference effects for QoS-aware clouds,” *Proceedings of the 5th European conference on Computer Systems* (EuroSys 10), 2010, pp. 237–250.
4. K. Gardener et al., “Reducing latency via redundant requests: Exact analysis,” *ACM SIGMETRICS*, vol. 43, no. 1, 2015, pp. 347–360.
5. J.F. Pérez et al., “Dual scaling vms and queries: Cost-effective latency curtailment,” *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems* (ICDCS 18), 2017, pp. 988–998.
6. J. Dean and L.A. Barroso, “The tail at scale,” *Communications of the ACM*, vol. 56, no. 2, 2013, pp. 74–80.

7. S. Melnik et al., "Dremel: Interactive analysis of web-scale datasets," *Proceedings of the 36th International Conference on Very Large Data Bases* (PVLDB 10), 2010, pp. 330–339.
8. Z. Qiu et al., "Cutting latency tail: Analyzing and validating replication without canceling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, 2017, pp. 3128–3141.
9. A. Vulimiri et al., "Low latency via redundancy," *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies* (CoNEXT 13), 2013, pp. 283–294.
10. G. Joshi, E. Soljanin, and G.W. Wornell, "Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing," *Efficient replication of queued tasks for latency reduction in cloud systems* (Allerton 15), 2015, pp. 107–114.
11. Z. Qiu and J.F. Pérez, "Evaluating the effectiveness of replication for tail-tolerance," *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (CCGrid 15), 2015, pp. 443–452.
12. K. Lee, R. Pedarsani, and K. Ramchandran, "On scheduling redundant requests with cancellation overheads," *Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing* (Allerton), 2015, pp. 99–106.
13. Z. Qiu, J. Pérez, and P. Harrison, "Variability-aware request replication for latency curtailment," *Proceedings of the 35th Annual IEEE International Conference on Computer Communications* (INFOCOM 16), 2016.
14. R. Birke et al., "sPARE: Partial replication for multi-tier applications in the cloud," *IEEE Transactions on Services Computing*, 2017; doi.org/10.1109/TSC.2017.2780845.
15. M. Fazio et al., "Open issues in scheduling microservices in the cloud," *IEEE Cloud Computing*, vol. 3, no. 5, 2016, pp. 81–88.
16. Z. Qiu and J.F. Pérez, "Enhancing reliability and response times via replication in computing clusters," *Proceedings of the 34th Annual IEEE International Conference on Computer Communications* (INFOCOM 15), 2015, pp. 1355–1363.

AUTHOR BIOS

Juan F. Pérez is an assistant professor at Universidad del Rosario, Colombia, Department of Applied Mathematics and Computer Science. He received a PhD degree in Computer Science from University of Antwerp, Belgium. His research interests center around the performance analysis of computer systems, especially on cloud and cluster computing and optical networking. Contact him at juanferna.perez@urosario.edu.co.

Lydia Y. Chen is a research staff member at the IBM Zürich Research Lab, Switzerland. She received a PhD degree in Operations Research and Industrial Engineering from the Pennsylvania State University. Her research interests include performance evaluation for datacenters and big data systems. She has served on several technical program committees in various performance and network conferences. She is an IEEE senior member. Contact her at yic@zurich.ibm.com.

Massimo Villari an associate professor of computer science at the University of Messina. His research interests include cloud computing, Internet of Things, big data analytics, and security systems. Villari has a PhD in computer engineering from the University of Messina. He's a member of IEEE and IARIA boards. Contact him at mvillari@unime.it.

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.

Blockchain: A Panacea for Healthcare Cloud-Based Data Security and Privacy?

Christian Esposito
University of Salerno

Alfredo De Santis
University of Salerno

Genny Tortora
University of Salerno

Henry Chang
University of Hong Kong

Kim-Kwang Raymond Choo
University of Texas at San Antonio

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

One particular trend observed in healthcare is the progressive shift of data and services to the cloud, partly due to convenience (e.g. availability of complete patient medical history in real-time) and savings (e.g. economics of healthcare data management). There are, however, limitations to using conventional cryptographic primitives and access control models to address security and privacy concerns in an increasingly cloud-based environment. In this paper, we study the potential to use the Blockchain technology to protect healthcare data hosted within the cloud. We also describe the practical challenges of such a proposition and further research that is required.

Healthcare is a data-intensive domain where a large amount of data is created, disseminated, stored, and accessed daily. For example, data is created when a patient undergoes some tests (e.g. computerized tomography or computerized axial tomography scans), and the data will need to be disseminated to the radiographer and then a physician. The results of the visit will then be stored at the hospital, which may need to be accessed at a later time by a physician in another hospital within the network.

It is clear that technology can play a significant role in enhancing the quality of care for patients (e.g. leveraging data analytics to make informed medical decisions) and potentially reduce costs by more efficiently allocating resources in terms of personnel, equipment, etc. For example, data

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

captured in paper form is hard to capture in systems (e.g. costly and data entry errors), costly to archive, and being available when needed. These challenges may lead to medical decisions not made with complete information, the need for repeated tests due to missing information or data being stored in a different hospital at a different state or country (at the expenses of increasing costs and inconvenience for the patients), etc. Due to the nature of the industry, ensuring the security, privacy, and integrity of healthcare data is important. This highlights the need for a sound and secure data management system.

HEALTH RECORDS IN ELECTRONIC FORMS AND HEALTH INFORMATION SYSTEMS

Generally, *Electronic Medical Records* (EMRs) contain medical and clinical data related to a given patient and stored by the responsible healthcare provider.[1] This facilitates the retrieval and analysis of healthcare data. To better support the management of EMRs, early generations of *Health Information Systems* (HIS) are designed with the capability to create new EMR instances, store them, and query and retrieve stored EMRs of interest.[2] HIS can be relatively simple solutions, which can be schematically described as a graphical user interface or a web service. These are generally the front-end with a database at the back-end, in a centralized or distributed implementation.

With patient mobility (both internally and externally to a given country) being increasingly the norm in today's society, it became evident that multiple stand-alone EMR solutions must be made interoperable to facilitate sharing of healthcare data among different providers, even across national borders, as needed. For example, in medical tourism hubs such as Singapore, the need for real-time healthcare data sharing between different providers and across nations becomes more pronounced.

To facilitate data sharing or even patient data portability, there is a need for EMRs to formalize their data structure and the design of HIS. *Electronic Health Records* (EHRs), for example, are designed to allow patient medical history to move with the patient or be made available to multiple healthcare providers (e.g. from a rural hospital to a hospital in the capital city of the country, before the patient seeks medical attention at another hospital in a different country).[3] EHRs have a richer data structure than EMRs. There have also been initiatives to develop HIS and infrastructures that are able to scale and support future needs, as evidenced by the various national and international initiatives such as the Fascicolo Sanitario Elettronico (FSE) project in Italy, the epSOS project in Europe, and an ongoing project to standardize sharing of EHRs.[4,5,6]

Recently, the pervasiveness of smart devices (e.g. Android and iOS devices and wearable devices) has also resulted in a paradigm shift within the healthcare industry.[7] Such devices can be user-owned or installed by the healthcare provider to measure the well-being of the users (e.g. patients) and inform/facilitate medical treatment and monitoring of patients. For example, there is a wide range of mobile applications (apps) in health, fitness, weight-loss, and other healthcare related categories. These apps mainly function as a tracking tool, such as registering user exercises/workouts, keeping the count of consumed calories, and other statistics (e.g. number of steps taken), and so on.

There are also devices with embedded sensors for more advanced medical tasks, such as bracelets to measure heartbeat during workouts, or devices for self-testing of glucose. For example, Leu and collaborators proposed a smartphone-based wireless body sensor network to collect user physiological data using body sensors embedded in a smart shirt.[8] The data (e.g. user's vital signs) can be continuously gathered and sent in real-time to a smart device, before being sent to a remote healthcare cloud for further analysis. Another example is Ambient Assisted Living solutions for healthcare designed to realize innovative telehealth and telemedicine services, in order to provide remote personal health monitoring.[9]

These developments have paved the way for *Personal Health Records* (PHR), where patients are more involved in their data collection, monitoring of their health conditions, etc, using their smart phones or wearable devices (e.g. smart shirts and smart socks).[10,11]

There are, however, a number of challenges associated with PHRs. For example, can we rely on the data collected by the patients themselves? Should the relevant healthcare providers certified data collected by the patients, and if so, how can this be done? Who should be legally liable for a misdiagnosis or delayed diagnosis, due to decisions being made on the data sent from the patient's device that is subsequently determined to be flawed or inaccurate (e.g. due to a malfunction sensor)?

Despite such challenges and potentially thorny legal issues, having a HIS based on an ecosystem of solutions that is able to seamlessly exchange data among themselves and provide the abstraction of a single health data storage for any given patient (e.g. physically distributed among multiple concrete software instances at multiple healthcare providers and mobile apps) will benefit all users, ranging from patients to healthcare providers to governments.

Cloud computing is a potential solution, due to the capability to support real-time data sharing regardless of geographical locations, to provide resource elasticity as needed, and to handle big data (e.g. hosting of big data analytical tools) to obtain useful insights from the analysis of big healthcare data for research and policy decision making.[12,13]

In Figure 1, we demonstrate how cloud help facilitate sharing of healthcare data among providers, supporting each provider in managing their data, providing a seamless way of exchanging and potentially certifying data between EHR and PHR, and providing a unified/comprehensive view of (the scattered) healthcare records for each patient. In other words, (federated) cloud computing can be used to interconnect the different healthcare providers and their PHR solutions, used by the providers to deal with any sudden or seasonal changes, and so on.

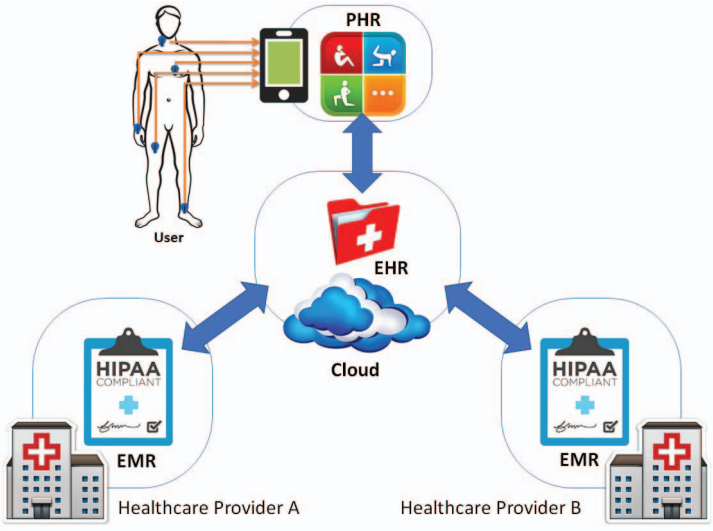


Figure 1. A conceptual cloud-based EMR/EHR/PHR ecosystem.

SECURITY AND PRIVACY

Healthcare data contain personal and sensitive information that may be attractive to cybercriminals. For example, cybercriminals seeking to benefit financially from the theft of such data may sell the data to a third-party provider, who may perform data analysis to identify individuals who may be uninsurable due to their medical history or genetic disorder. Such data would be of interest to certain organizations or industries.

Therefore, ensuring the security of the EMR/EHR/PHR ecosystem and the underlying systems and components that form the ecosystem is crucial, yet challenging due to the interplay and complexity between the systems and components. Moreover, the privacy and integrity of healthcare data must be protected not only from external attackers, but also from unauthorized access attempts from inside the network or ecosystem (e.g. employee of the healthcare provider, or cloud

service provider). The attacks (e.g. leakage or modification of data) can be intentional and unintentional, and organizations may be penalized or held criminally liable for such incidents, for example under the Health Insurance Portability and Accountability Act.

How to secure EMR/EHR/PHR ecosystem and ensure privacy and integrity of the data is an active research area. Approaches include using cryptographic primitives, such as those based on public key infrastructure and public clouds to ensure data confidentiality and privacy.[14] For example, data is encrypted prior to outsourcing to the cloud. However, this limits the searchability of the data, in the sense that healthcare providers have to decrypt the (potentially big) data prior to searching on the decrypted data, resulting in increases in time and costs for the data retrieval and diagnosis (e.g. download, decrypt, and search).[15]

Access control models have also been used to regulate and limit access to the data, based on predefined access policies.[16] Such models can be particularly effective for external attacks, but are generally ineffective against internal attackers as they are likely to be authorized to access the data. There have also been approaches to integrate access control with some cryptographic primitives, such as attribute-based encryption.[17]

BLOCKCHAIN TO THE RESCUE?

There has been recent interest in utilizing blockchain (made popular by the successful Bitcoin) in the provision of secure healthcare data management.[18,19,20] Broadly speaking, blockchain is a technology able to build an open and distributed online database, which consists a list of data structures (also known as blocks) that are linked with each other (i.e. a block points to the following one, hence the name blockchain). These blocks are distributed among multiple nodes of an infrastructure, and are not centrally stored. Each block contains a timestamp of its production, the hash of the previous block and the transaction data, and in our context, a patient's healthcare data and the healthcare provider information.

Figure 2 describes our conceptual blockchain-based EMR/EHR/PHR ecosystem. Specifically, when new healthcare data for a particular patient is created (e.g. from a consultation, and medical operation such as a surgery), a new block is instantiated and distributed to all peers in the patient network. After a majority of the peers have approved the new block, the system will insert it in the chain. This allows us to achieve a global view of the patient's medical history in an efficient, verifiable, and permanent way. If the agreement is not reached, then a fork in the chain is created and the block is defined as an orphan and does not belong to the main chain. Once the block has been inserted into the chain, the data in any given block cannot be modified without modifying all subsequent blocks. In other words, modification can be easily detected. As block content is publicly accessible, healthcare data needs to be protected prior to the data being in the block (e.g. obfuscated and perhaps, encrypted).

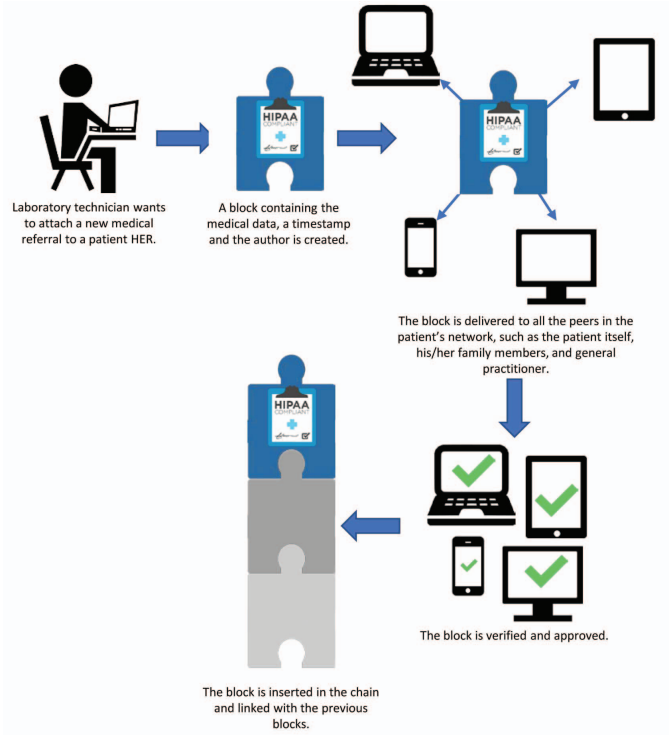


Figure 2. A conceptual blockchain-based EMR/EHR/PHR ecosystem.

Conceptually, blockchain is secure by design that provides the capability to achieve decentralized consensus and consistency, and resilience to intentional and/or unintentional attacks. Key benefits of deploying a blockchain in our approach are as follows:

1. Agreement can be reached without the involvement of a trusted mediator; thus, avoiding a performance bottleneck and a single point of failure;
2. Patients have control over their data;
3. Medical history as a blockchain data is complete, consistent, timely, accurate, and easily distributed; and
4. Changes to the blockchain are visible to all members of the patient network, and all data insertions are immutable. Also, any unauthorized modifications can be trivially detected.

As with any security solutions, there are limitations associated with a blockchain-based approach that need to be carefully studied. For example, blockchain technology can be somewhat disruptive and requires a radical rethink and significant investment in the entire ecosystem (e.g. replacement of existing systems and redesigning of business processes). In other words, before taking the plunge, healthcare providers particularly publicly funded providers will need to undertake a cost benefit analysis to understand the return on investment and any potential implications (e.g. legal and financial). For example, the same record can reside in multiple nodes of the network, located in different countries with different privacy and data protection requirements (e.g. EU and US).

CHALLENGES

While data integrity and distributed storage/access of blockchain offer opportunities for healthcare data management, these same features also pose challenges that need further study.[21]

The strong data integrity feature of blockchain results in immutability that any data, once stored in blockchain, cannot be altered or deleted. However, if the record is healthcare data, then such

personal data would come under the protection of privacy laws, many of them would not allow personal data to be kept perpetually—Article 17 of the soon-enforceable General Data Protection Regulation in the EU has strengthened the rights of individuals to request personal data to be erased. One of the principles of the Organization for Economic Cooperation and Development privacy guideline, on which many data protection laws are based, provides the right-to-erasure to individuals. Given the sensitivity of healthcare data, anyone planning to use blockchain to store them cannot ignore this legal obligation to erase personal data if warranted.

Another practical issue is on how fit it is for blockchain to store healthcare data. Blockchain was originally designed to record transaction data, which is relatively small in size and linear. In other words, one only concerns itself about whether the current transaction can be traced backwards to the original "deal". Healthcare data, such as imaging and treatment plans, however, can be large and relational that requires searching. How well blockchain storage can cope with both requirements is currently unclear.

In order to deal with these challenges, many have suggested the notion of off-chain storage of data, where data is kept outside of blockchain in a conventional or a distributed database, but the hashes of the data are stored in the blockchain. This is said to be the best of both worlds, as healthcare data is stored off-chain and may be secured, corrected, and erased as appropriate. At the same time, immutable hashes of the healthcare data are stored on-chain for checking the authenticity and accuracy of the off-chain medical records.

This idea, however, is not without potential challenges. With the tightening of data protection laws around the world and the attempts by privacy commissioners to regard metadata of personal data as personal data, it may not be very long that hashes of personal data are considered as personal data; then the whole debate of whether blockchain is fit to store personal data may start all over again.

REFERENCES

1. M. Steward, "Electronic Medical Records," *Journal of Legal Medicine*, vol. 26, no. 4, 2005, pp. 491–506.
2. R. Hauxe, "Health Information Systems—Past, Present, Future," *Int'l Journal of Medical Informatics*, vol. 75, no. 3–4, 2006, pp. 268–281.
3. K. Häyrinena et al., "Definition, Structure, Content, Use and Impacts of Electronic Health Records: A Review of the Research Literature," *Int'l Journal of Medical Informatics*, vol. 77, no. 5, 2008, pp. 291–304.
4. M. Ciampi et al., "A Federated Interoperability Architecture for Health Information Systems," *Int'l Journal of Internet Protocol Technology*, vol. 7, no. 4, 2013, pp. 189–202.
5. M. Moharra et al., "Implementation of a Cross-Border Health Service: Physician and Pharmacists' Opinions from the epSOS Project," *Family Practice*, vol. 32, no. 5, 2015, pp. 564–567.
6. S.H. Han et al., "Implementation of Medical Information Exchange System Based on EHR Standard," *Healthcare Informatics Research*, vol. 16, no. 4, 2010, pp. 281–289.
7. D. He et al., "A Provably-Secure Cross-Domain Handshake Scheme with Symptoms-Matching for Mobile Healthcare Social Network," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, 2016; doi.org/DOI: 10.1109/TDSC.2016.2596286.
8. F.Y. Leu et al., "A Smartphone-Based Wearable Sensors for Monitoring Real-Time Physiological Data," *Computers and Electrical Engineering*, 2017.
9. M. Memon et al., "Ambient Assisted Living Healthcare Frameworks, Platforms, Standards, and Quality Attributes," *Sensors*, vol. 14, no. 3, 2014, pp. 4312–4341.
10. P.C. Tang et al., "Personal Health Records: Definitions, Benefits, and Strategies for Overcoming Barriers to Adoption," *Journal of the American Medical Informatics Assoc.*, vol. 13, no. 2, 2006, pp. 121–126.
11. S. Marceglia et al., "A Standards-Based Architecture Proposal for Integrating Patient mHealth Apps to Electronic Health Record Systems," *Applied Clinical Informatics*, vol. 6, no. 3, 2015, pp. 488–505.

12. A. Mu-Hsing Kuo, "Opportunities and Challenges of Cloud Computing to Improve Health Care Services," *Journal of Medical Internet Research*, vol. 13, no. 3, 2011.
13. V. Casola et al., "Healthcare-Related Data in the Cloud: Challenges and Opportunities," *IEEE Cloud Computing*, vol. 3, no. 6, 2016, pp. 10–14.
14. S. Nepal et al., "Trustworthy Processing of Healthcare Big Data in Hybrid Clouds," *IEEE Cloud Computing*, vol. 2, no. 2, 2015, pp. 78–84.
15. G.S. Poh et al., "Searchable Symmetric Encryption: Designs and Challenges," *ACM Computing Surveys*, vol. 50, no. 3, 2017.
16. Q. Alam et al., "A Cross Tenant Access Control (CTAC) Model for Cloud Computing: Formal Specification and Verification," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, 2017, pp. 1259–1268.
17. M. Li et al., "Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 3, 2016, pp. 2084–2123.
18. F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, 2016, pp. 2084–2123.
19. A. Azaria et al., "MedRec: Using Blockchain for Medical Data Access and Permission Management," *Proceedings of the 2nd Int'l Conference on Open and Big Data* (OBD 16), 2016, pp. 25–30.
20. J. Zhang, N. Xue, and X. Huang, "A Secure System for Pervasive Social Network-Based Healthcare," *IEEE Access*, vol. 4, 2016, pp. 9239–9250.
21. J. McKinlay et al., "Blockchain: Background, Challenges and Legal Issues," *DLA Piper Publications*, 2016; doi.org/https://www.dlapiper.com/en/uk/insights/publications/2017/06/blockchain-background-challenges-legal-issues/.

ABOUT THE AUTHORS

Christian Esposito is an adjunct professor at the University of Naples "Federico II," where he received his PhD in computer engineering and automation. He is also a research fellow at the University of Salerno, Italy. Esposito's research interests include reliable and secure communications, middleware, distributed systems, positioning systems, multiobjective optimization, and game theory. Contact him at christian.esposito@dia.unisa.it.

Alfredo De Santis is a professor of computer science and the department director at the University of Salerno, where he received a degree in computer science. His research interests include data security, cryptography, digital forensics, communication networks, data compression, information theory, and algorithms. Contact him at ads@unisa.it.

Genny Tortora is a full professor of computer science and was dean of the faculty of Mathematical, Natural, and Physical Sciences from 2000 to 2008 at the University of Salerno, where she received a degree in computer science. Her research interests include software-development environments, visual languages, geographical information systems, biometry, and virtual reality. Contact her at tortora@unisa.it.

Henry Chang is an adjunct associate professor at the Department of Law in the University of Hong Kong. His research interests include technological impact on privacy. Chang is a fellow of the British Computer Society and a member of the Hong Kong/Guangdong ICT Expert Committee on Cloud. Contact him at hcychang@hku.hk.

Kim-Kwang Raymond Choo is the holder of 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.

THORIN: an Efficient Module for Federated Access and Threat Mitigation in Big Stream Cloud Architectures

Luca Davoli
University of Parma

Laura Belli
University of Parma

Luca Veltri
University of Parma

Gianluigi Ferrari
University of Parma

In order to make cloud services attractive for several IT organizations, it is necessary to provide access control and implement safe and reliable mechanisms of Identity and Access Management (IAM). This article focuses on security issues and challenges in the design and implementation of cloud architectures and, in particular, for the management of Big Stream applications in Internet of Things (IoT) scenarios. The proposed work introduces a new set of modules allowing a federated access control policy for cloud users. We present an analysis of possible threats and attacks against the proposed Big Stream platform, investigating the system performance in terms of detection and elimination of malicious nodes. In particular, we propose a new module, denoted as Traffic Handler Orchestrator & Rapid Intervention (THORIN), which is very efficient in counteracting botnet-based threats.

Cloud computing can be described as an infrastructure computing paradigm which consists of a collection of interconnected computing nodes, servers, and other hardware, as well as software services and applications, that are rapidly and dynamically provisioned among end users with minimal management efforts. Services are delivered over the Internet, private networks, or through their combination, and are accessed on the basis of their availability, performance, and Quality of Service (QoS) requirements.[1]

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

Nowadays, cloud computing is constantly growing and so are concerns about data security and privacy that give rise to additional cloud-specific vulnerabilities (i.e., the possibility that a component will be unable to resist against the actions of a threat agent). Following the public, shared, and virtualized nature of the cloud computing paradigm, the migration of assets (e.g., data and applications) from an administrative control environment to a shared environment where many users are co-located, highly increases security concerns.[2–3]

From an infrastructure point of view, the services offered by the cloud paradigm are automatically managed following a multi-tenant model, exploiting virtualization technologies on one or more physical servers. Similarly to other paradigms, cloud computing needs to manage the digital identity of its users. A possible solution is represented by Identity Management (IdM),[4] corresponding to a set of capabilities (such as maintenance, administration, authentication and policy enforcement) used to ensure identity information and guarantee security. According to this perspective, since all data are outside the domain of the consumer, cloud computing opens new scientific challenges about access control, security, and privacy. In order to make cloud services attractive for organizations, it is first necessary to provide access control and implement safe and reliable mechanisms of Identity and Access Management (IAM). Moreover, some IAM systems allow management of federations, which are groups of organizations establishing trust among themselves in order to have safe business cooperation and adopt identities denoted as federated. In this type of IAM systems, once a user is authenticated by an organization of the federation, he/she can use all its services as well as those of another federated organization, without the need to repeat the process of authentication. This optional behavior is not trivial, because it requires interoperability between different systems and security technologies. In fact, each organization could have specific authentication and IAM mechanisms, making interoperability challenging.

Another relevant trend rapidly growing in recent years is the Internet of Things (IoT), corresponding to a worldwide "network of networks" involving billions of different devices (nodes) connected to the Internet.[5] The IoT allows new forms of interaction among things and people, generating huge amounts of data, which can be processed and employed to build services for consumers in several scenarios (e.g., Smart Cities, monitoring and surveillance applications, e-health). Due to its scalability, robustness and cost-effectiveness, the cloud is, on one hand, the natural collection environment for data retrieved by IoT nodes; and, on the other hand, the ideal service provider for consumers' applications.

In our previous work,[6] we proposed a graph-based cloud architecture for the IoT, describing in detail the modules of the framework and the technologies employed for an open source implementation. This preliminary work does not take into account any aspect related to security. In another work,[7] we introduced new modules (denoted as In-Graph Security (IGS) and Outdoor Front-end Security (OFS)) to the architecture, aiming at increasing the platform security. In particular, we addressed the problem of "core" security inside the graph.[7]

We extend this platform architecture here to take into account also the access control problem (or "outer security") and the management of malicious nodes in the graph. In particular, we propose a new module, denoted as "Traffic Handler Orchestrator & Rapid Intervention" (THORIN), which is very efficient in counteracting botnet-based threats. Our results show that the proposed architecture can efficiently react against attacks of a malicious node, "sanitizing" the graph and removing compromised nodes in a transparent way, while other nodes remain unaware of the changes occurred.

RELATED WORK

In order to allow access control in open environments, IdM can be implemented in different ways: (i) in-house implementation: identities are issued and managed by single companies, which have complete responsibility on them; (ii) managed implementation: the Identity-as-a-Service (IDaaS) concept is adopted and IdM is outsourced to external companies. Regardless of the chosen configuration, an IdM System (IMS) controlling the platform identities always involves three main types of entities: (i) the user; (ii) the Identity Provider (IdP), responsible for issuing and managing user identities and credentials; and (iii) the Service Provider (SP), responsible for

providing services based on users' identities/attributes. An IMS must always provide the following functions: (i) provisioning of identities related to the different types of accounts adopted by the organization (e.g., administrator, IT consultant, developer, end user); (ii) authentication and authorization functions, identifying individuals through various mechanisms (e.g., username/password, X.509 certificates, OTP, biometrics) and providing them different access levels for different operations within a computing infrastructure; (iii) grouping SPs into a federation and, then, establishing a trust that allows sharing of identities' information among them.

Even if the topic is not fully explored, some works have recently appeared in the literature to address the problem of applying security mechanisms to the IoT-cloud environment. The work of Lake and colleagues[8] analyzes security issues in an IoT-based e-health scenario, providing a framework mainly focused on vulnerabilities associated with: (i) endpoint access; (ii) cloud services; and (iii) partners and providers. The work of Suciu and colleagues[9] explores a Smart Cities scenario, focusing on the problem of securing cloud APIs for sensing and actuation in an open sensor platform. Regardless of the selected use case, the most relevant technologies employed for implementing an IMS can be summarized as follows.

- Active Directory (AD): a directory service created by Microsoft for Windows domain networks.
- Security Assertion Markup Language (SAML): an XML-based open standard for exchanging authentication and authorization data between security domains, i.e., between an IdP and a SP.
- Single Sign-On (SSO): a mechanism for access control of multiple (related, but independent) software systems, in which a user logs in only once and gains access to all related systems without the need to log in again.

Storing and managing the identities present crucial security concerns for cloud providers, because stored information can be tampered or modified by malicious or unauthorized users. Several technologies implement federated identity, such as the Shibboleth system.[10] Shibboleth is a widely employed Authentication and Authorization Infrastructure (AAI) based on SAML that allows to create a safe structure that simplifies the management of identities and provides the user a SSO layer for different organizations belonging to the same federation, in which identity information is shared. A Shibboleth system is built on two main entities, the IdP and the SP, and on an optional one, the Where Are You From (WAYF) module. The Shibboleth IdP manages users' authentication, maintaining credentials and attributes. The SP is located where the resources are stored and accessed by the user. The optional WAYF component allows an association between a user and multiple organizations. When trying to access a resource, the user is forwarded to an interface that asks him/her to choose the organization which he/she belongs to; after this choice, the user is redirected to his/her correct IdM interface to start the authentication process.

The growing diffusion of IoT applications, based on large-scale sensor networks, has spurred research efforts in studying data streams processing and distribution mechanisms. However, several aspects related to data streams' security still need to be investigated. Puthal and colleagues[11] address the end-to-end data stream security problem (e.g., integrity and authenticity) for risk-critical applications scenarios. Their proposed approach, called dynamic key length-based secure framework (DLSeF) is based on symmetric key cryptography, with a common shared key that is generated by exploiting synchronized prime number. The framework aims at avoiding excessive communication messages for the rekey process between data sources and the centralized collection point, called Data Stream Manager (DSM). By reducing the overall communication overhead, the DLSeF module is able to perform all security verification processing activities on-the-fly, avoiding data stream buffering. This solution is suitable for stream management applications where the both the endpoints of the communication are known, and cannot be adopted when data streams can be dynamically combined, without fixed source and destination points.

Another relevant issue is given by the authentication and privacy problem in the continuous sensing scenario. Emerging technologies, such as Google Glass and Microsoft Kinect, rely on continuous recording of audio or video to provide services to users (i.e., voice command or gestures); however, these capabilities raise serious privacy and security concerns. Roesner and colleagues[12] propose a framework for controlling access to sensor data on multi-application

continuous sensing platforms. This approach is based on world-driven access control, which allows real-world objects to explicitly specify access policies. In the authors' vision, users themselves do not specify any policy, but devices automatically detect policies broadcasted by real world's objects through a particular certificate called passport. The proposed approach can be extended and is suitable to scenarios where there is an explicit interaction between humans and things. It does not cover other possible applications related to the IoT scenarios.

THE BIG STREAM CLOUD ARCHITECTURE: CONCEPT AND BASIC IMPLEMENTATION

Several relevant IoT applications (e.g., alerting and monitoring systems[13]) require real-time performance or, at least, a predictable (and consistent) latency. On one hand, the large number of IoT data sources globally generate data at a high rate (even though a single IoT service generates a limited amount of data); on the other hand, the low-latency constraints call for innovative cloud architectures, able to efficiently handle such massive amount of information. A possible solution is given by Big Data approaches, which address the need to process extremely large amounts of heterogeneous data for various purposes. However, these techniques typically have an intrinsic inertia and focus on the data itself, rather than providing real-time processing and dispatching. For these reasons, Big Data approaches might not be the right solution to manage the dynamicity of IoT scenarios and, thus, Belli and colleagues[6] proposed a shift from the Big Data paradigm to a new one, denoted as "Big Stream." A cloud-oriented graph-based architecture, explicitly designed to fit Big Stream IoT scenarios, has also been proposed and implemented.[6–7] In order to minimize the delay between the instant of raw data generation (e.g., at the sensors) and the instant at which properly processed information is provided to the final consumer, the proposed platform adheres to the publish/subscribe model and is based on the concept of "listener." More in detail, the fundamental components of the graph-based cloud architecture are the following.

- Nodes: processing units which process incoming data, not directly linked to a specific task. A graph node can be a listener of one or more streams and a publisher of a new stream for other nodes. Nodes in the graph are logically organized in layers, characterized by an increasing degree of complexity.
- Edges: streams linking together various nodes, allowing complex processing operations.

Although the Big Stream architecture outlined above can be deployed on a single server, exploiting it on the cloud allows to improve the system's scalability and to manage the workload with a huge number of data sources and processing nodes. Another important benefit, brought by the use of cloud, is that it provides a common platform in which data streams can be shared among several actors (e.g., IoT data sources, developers or consumers), in order to build useful services for different consumers. Therefore, this architecture is mainly intended for developers interested in building applications based on data originated by IoT networks, with real-time constraints and low overhead. The platform provides the following services for developers: (i) processing nodes upload/deletion; (ii) internal stream status; (iii) external data source upload/deletion (i.e., a new IoT provider). Developers authenticated on the platform can thus customize paths in the graph through the definition and deployment of new processing nodes. It is important to observe that, by accessing the Big Stream architecture, each developer can operate on data streams coming from IoT networks which he/she does not own.

SECURITY IN THE BIG STREAM ARCHITECTURE

In the proposed Big Stream architecture, security was originally implemented in two clearly separated levels[7]: (i) inner security, through the IGS module, and (ii) outer security, with the OFS module. As shown in Figure 1, the IGS module manages the secured streams, defining a set of rules indicating which actors are authorized to send/receive data to/from each single processing node. To accomplish this task, the IGS module works with the Application Register (AR) module, which is composed by the following components (shown in Figure 1):

- the Graph State Database (GSDB) and the Node Registration and Queue Manager (NRQM) modules, which cooperate to manage the authorization policies adopted by the graph nodes;
- the Policy Manager and Storage Validator (PMSV) module;
- the Graph Rule Authorization Notifier (GRAN) module; and
- the Persistent Security Storage Container (PSSC) module.

The OFS module operates at the borders of the platform, after receiving input data from IoT networks, as well as before pushing out streams to final consumers. OFS has a crucial role, since it authorizes the interactions between the external entities and the frontier of the cloud platform. Since OFS should support both "open" and "secured" communications through different protocols (e.g., HTTP, CoAP and MQTT), in this module some security mechanisms have to be implemented (e.g.: at the application layer through S/MIME or OAuth; at the transport layer through TLS/DTLS). The IGS module allows to define paths in which some segments are "secured" and some others are "public." Beside the basic functionalities outlined above, security features in the core part of the platform itself need to be improved, adopting policies (e.g., access control and federated authentication) like those explained in the following and, in particular, those provided by the THORIN module.

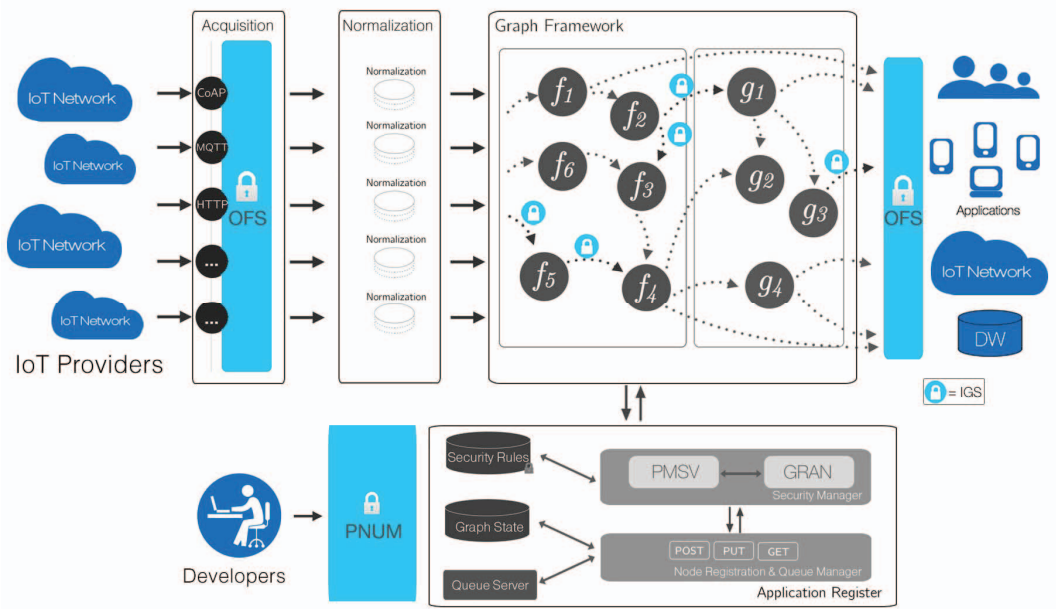


Figure 1. Details of the security modules embedded in the proposed Big Stream architecture.

Federated Access Control in Big Stream

Following the SaaS paradigm, the Big Stream architecture gives external providers the possibility to upload into the graph custom nodes containing executable (software) code. Although this functionality is an added value for the platform, it clearly opens several security vulnerabilities. A possible countermeasure is the adoption of an encrypted channel on which a developer can upload its nodes via a transfer protocol (e.g., using SFTP). Another one is represented by a federated authentication system. This approach allows to create a federation of trusted entities aiming at accessing to the Big Stream platform, through a federated access paradigm (e.g., Shibboleth) and to share their own nodes. Both these solutions are handled by the Processing Node Upload Manager (PNUM) and are depicted in Figure 2.

The adoption of a federation-based access paradigm allows to get rid of the need to store and maintain an account for each developer, reducing system vulnerabilities and making the architecture more scalable and accessible through the Web. This type of authentication mechanism can

also be adopted for the output stage of the Big Stream platform, in order to authenticate a final consumer requesting a stream coming from the higher layer nodes. By exploiting the capabilities of the proposed Big Stream platform, together with its acquisition module, it is possible to define policies to maintain a sufficiently high QoS for external providers. This can be carried out through the arrangement of the following special communication channels for secured streams.

- Reservation of acquisition nodes, providing a clone for each requested communication protocol (e.g., CoAP, HTTP): this is possible in a limited time (e.g., on the order of the startup time of a Java process, between 1 s and 5 s), because of the highly dynamic nature of the system and the specificity of the nodes that have to be cloned.
- Transfer of the processing nodes, that have to be deployed into the platform, encapsulating them into a secure communication tunnel (e.g., through IPSec) after federated authentication.
- Encryption of the transferred data through a public/private encryption mechanism, using previously negotiated keys between the Big Stream platform and the IdM of the organization which the external provider belongs to.

As previously mentioned, the same solution can be adopted at the output stage of the Big Stream platform, if a final consumer needs some additional streams. These security measures allow to control and manage the access requests, denying the injection of malicious nodes. Moreover, in the case of heavy attacks (e.g., if the platform is under requests bombing), the acquisition nodes at the input stage of the platform can be replaced in a limited time (e.g., on the order of the startup time of a Java process, between 1 s and 5 s) and the authorized traffic hijacked to the newly activated nodes.

There are several aspects that make the proposed Big Stream platform adhering to the cloud paradigm. First, it can be deployed on different commercial cloud services (e.g., Amazon EC2 and Microsoft Azure). Moreover, the internal graph organization platform can be assimilated to a wide cloud interconnection of systems, in which each single cloud entity corresponds to a single processing node. However, the Big Stream platform is vulnerable to a botnet-based attack, in which a multitude of systems get infected and become zombie machines, remotely controlled by Command-and-Control (C&C) servers, often hidden behind a Tor-based network. The success of this type of attack can provide the attackers with a high firepower, as the number of nodes running inside the graph of the cloud-oriented Big Stream architecture is large. For this reason, it is fundamental to deny the attackers any possibility (i) to catch the command of any processing node and (ii) to inject malicious nodes exploiting the cohesion of the Big Stream platform.

In order to protect the Big Stream platform against this botnet-based threat, we now propose a new security module, denoted as Traffic Handler Orchestrator & Rapid Intervention (THORIN), deployed in the graph platform and connected to the Application Register module.

THORIN

THORIN monitors the traffic on the overall platform, analyzing in real-time the behavior of the different components, with the help of heuristics and optimization techniques. The joint action of Application Register and THORIN modules aims at:

- controlling the interactions between the external providers and the acquisition front-end nodes of the Big Stream platform;
- analyzing the behavior of the internal graph of the platform, looking for suspicious activities carried out by processing nodes;
- reacting to malicious activities and protecting the other active nodes, allowing them to safely continue their operations.

A cloud-oriented infrastructure has to provide countermeasures against security threats related to both the interactions with external entities and the inner communications among graph processing nodes. Concerning the outside interactions, a security challenge is represented by the

need to guarantee access control, providing services only to authorized entities (i.e., IoT providers, developers, final consumers). This can be done, as previously highlighted, in a centralized way, as well as by delegating to a trusted third-party entity the management of the security policies (such as that adopted in our proposed platform, that allows to increase the number of collaborating entities and, at the same time, avoids the centralization of security management). In particular, THORIN implements a Shibboleth instance that allows different providers (linked to a wide federation) to connect and provide their processing nodes to the Big Stream platform. Considering communication between processing nodes, a challenge faced by the proposed architecture is its reactivity in the presence of security threats. Once authenticated, a federated user can use the architecture for its purposes: the presence of malicious users is thus a tangible security issue and proper considerations are needed to evaluate the security degree of the proposed platform. An authenticated malicious user can affect the system in different points of the infrastructure, as shown in Figure 2.

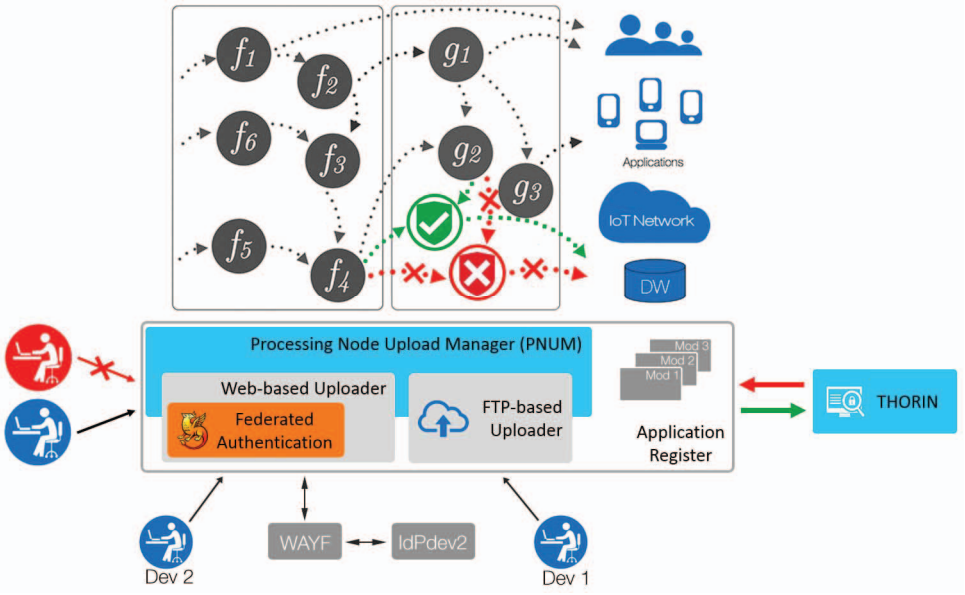


Figure 2. Big Stream platform's behavior to counteract internal and external security threats.

Thanks to its internal graph structure and the presence of THORIN, the Big Stream platform can withstand the attacks outlined in the previous paragraph better than a "standard" cloud-oriented architecture. We can summarize the motivations behind this robustness as follows.

First, being based on a publish/subscribe paradigm, the Big Stream implementation has some communication entities (namely, the "brokers") that can be clustered to improve the redundancy of functionalities and the system resilience. An attack against one of these brokers may be thwarted in a short time, removing it from the cluster and replacing it with a new one. In case of a "standard" cloud infrastructure, the reaction time increases, as a new virtual machine (VM) must be activated to replace the affected one that worked as broker before the attack, and its activation time is not negligible (in the order of hundreds of seconds). In particular, THORIN contains a particular Java module that replaces the broker under attack with a new (honest) one.

Second, when a federated node is recognized to have malicious behaviors (i.e., acting as DDoS attacker or as e-mail spammer), the THORIN module immediately reacts as follows: (i) shuts off the external provider of malicious data; (ii) removes it from the front-end interface; (iii) reports its behavior to the proper IdM; and (iv) prevents it from connecting to the acquisition interfaces of the Big Stream platform. This is shown on the left side of Figure 2, in which a malicious developer is banned from the architecture. Conversely, a "basic" cloud-oriented infrastructure requires the adoption of an external control software that handles all these operations. In particular, when a federated entity is admitted to be a provider/consumer of the Big Stream platform, THORIN transparently creates a hidden channel h_x (e.g., a RabbitMQ queue) that is linked and

used to communicate with the federated entity. In case of malicious behavior, THORIN (through its internal Java modules) immediately shuts-off and disconnects the federated entity and, through h_x, notifies the federated entity about the detected problem. Then, it avoids further connections from the ex-federated entity and deletes h_x, saving computational resources.

Finally, handling the security into the inner layers of the infrastructure, THORIN can intercept malicious behavior of the processing node (i.e., a zombie device in a botnet-based scenario) and can recover and restore, in a short time (between 1 s and 20 s), the full functionalities of the graph. THORIN does so through the following steps:

1. Isolating the internal malicious processing nodes, which are remotely managed by a malicious controller (e.g., a C&C server).
2. Protecting the other running nodes, allowing them to continue their execution safely.
3. Changing the routing keys associated with the streams under attack, to isolate the malicious nodes in the graph, denying the reception of further information by them and preventing them from sending malicious processed data to the broker working in their current graph layer (e.g., the red node in Figure 2).
4. Replacing the isolated malicious nodes with new clones (e.g., the green node in Figure 2), in order to reestablish the correct operational behavior of the internal graph.

Conversely, a "basic" cloud-oriented architecture needs more time than the one requested by the Big Stream platform, to power off the VM hit by a security attack and to replace it with a newly activated one, on the order of more than a minute.[14]

Regarding the monitoring activity of the inner layers, THORIN operates according to the following heuristic principles, namely: (i) running an internal Java module that aggregates and performs statistics on data flows, as well as (ii) interacting with external heuristic services and (iii) analyzing behaviors and comparing URLs used by inner processing nodes with publicly available dangerous blacklists (e.g., spam URLs, malware and ransomware URLs, darknet onion-like URLs, etc.).

SYSTEM EVALUATION ON A REAL USE-CASE

In order to prove the efficiency of the inner security approach guaranteed by the proposed architecture, the time required to isolate a malicious node (e.g., a node that tries to inject spam in a public news feed system) can be evaluated. Considering a real use-case, we assume that THORIN, during its runtime streams analysis, identifies a malicious Java node, denoted as M, injecting spam news containing dangerous URLs (e.g., targeting sites in the darknet). The node M has n Java "listener" nodes, each of them receiving q different streams produced by the malicious node. After the node M has been identified, THORIN should remove and replace a total of $q \times n$ streams, which, in our implementation, correspond to RabbitMQ queues. In Figure 3, the time (dimension: [s]) required to isolate the node M and its q streams by the other safe Java nodes $\{f_j\}_{j=1}^{n}$ is shown as a function of n, considering various values of q (from 1 stream/node to 50 streams/node). In the same figure, the reference attacked portion of the graph is shown. The obtained results show that, in the worst case, with 2500 streams to be shut down (namely, $n = 50$ listening nodes with $n = 50$ streams/node), the system reacts in less than 17 seconds.

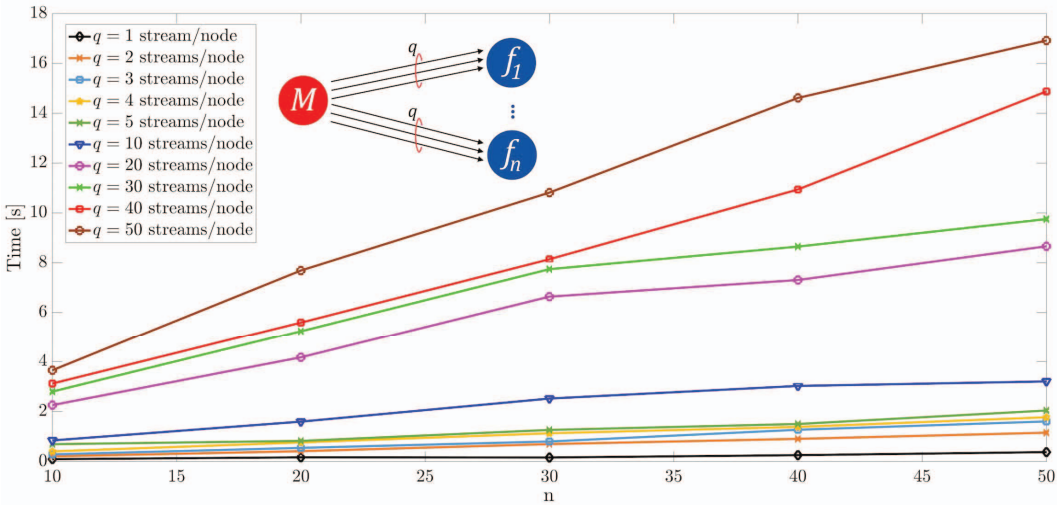


Figure 3. Performance results (in terms of reaction time) emulating an attack conducted by a malicious node *M* that affects *n* different listening nodes, each of them accepting incoming data on *q* streams.

The different steps involved in the deletion of the malicious node *M* by THORIN are the following:

1. isolating the malicious Java node *M*, removing its input/output RabbitMQ queues from the platform;
2. generating and binding new incoming RabbitMQ queues for the listener Java nodes $\{f_j\}_{j=1}^{n}$ to their proper RabbitMQ brokers, restoring the graph topology;
3. updating the modified queues details, related to Java nodes $\{f_j\}_{j=1}^{n}$ and to the malicious node *M*, into the SQL-based GSDB module, to maintain a consistent representation of the graph;
4. notifying each sanitized Java node $\{f_j\}_{j=1}^{n}$ of the updated consistent state, through reserved RabbitMQ queues exclusively binded between the same node and the Application Register module.

It is important to underline that: (i) the malicious Java node *M* is immediately isolated after step 1, executed in a very short time (in our evaluation, unbinding a RabbitMQ queue requires tens of milliseconds), while the rest of the time is needed to update the state of the graph in a consistent way; and (ii) the other Java processing nodes are unaware of the occurred changes, since all the previous steps will be orchestrated by the THORIN module and executed in a totally transparent way.

CONCLUSIONS AND FUTURE WORK

In this work, we have analyzed security issues and access control policies in cloud architectures, with particular attention to the management of Big Stream applications in IoT scenarios. We have defined the characteristics of the Big Stream paradigm and described the details of a recently proposed graph-based Big Stream architecture for IoT. The earlier implementation of the Big Stream architecture already takes into account inner security through the use of specific modules (with IGS, PMSV, GRAN, and PSSC modules). The current work has focused on the architecture's outer security and access control, introducing a new module, denoted as THORIN, which allows a federated access control policy for cloud users. An analysis of possible challenges and attacks has highlighted that the proposed architecture implementation, thanks to its graph-based structure and the consequent introduction of the THORIN module, can manage and

solve malicious attacks more efficiently than other cloud applications. This has been verified by experimental results.

An alternative approach to guarantee inner security is represented by the adoption of paradigms that act directly on the information itself. A relevant example is given by Information-Centric Networking (ICN), in which the location of a specific content (i.e., origin, destination, or storage position) is representative of the content itself.[15] In our Big Stream architecture, ICN can allow single processing nodes to retrieve and authenticate the information regardless of knowing where it comes from and how it has been transported. The use of an ICN approach to further secure the proposed THORIN-based architecture is currently under study.

REFERENCES

1. P. M. Mell and T. Grance, "SP 800-145. The NIST Definition of Cloud Computing," Tech. Rep., 2011. doi:10.6028/NIST.SP.800-145.
2. N. Kshetri, "Privacy and security issues in cloud computing: The role of institutions and institutional evolution," Telecommunications Policy, vol. 37, no. 4, pp. 372–386, May 2013. doi:10.1016/j.telpol.2012.04.011.
3. M. Ali et al., "Security in cloud computing: Opportunities and challenges," Information Sciences, vol. 305, pp. 357–383, June 2015. doi:10.1016/j.ins.2015.01.025.
4. M. A. P. Leandro et al., "Multi-Tenancy Authorization System with Federated Identity for Cloud-Based Environments Using Shibboleth," 2012, pp. 88–93.
5. S. Cirani et al., "A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things," IEEE Internet of Things Journal, vol. 1, no. 5, pp. 508–521, October 2014. doi:10.1109/JIOT.2014.2358296.
6. L. Belli et al., "An Open-Source Cloud Architecture for Big Stream IoT Applications," in Interoperability and Open-Source Solutions for the Internet of Things. Springer, January 2015, pp. 73–88. doi:10.1007/978-3-319-16546-2_7.
7. L. Belli et al., "Applying Security to a Big Stream Cloud Architecture for the Internet of Things," International Journal of Distributed Systems and Technologies (IJDST), vol. 7, no. 1, pp. 37–58, January 2016. doi:10.4018/IJDST.2016010103.
8. D. Lake et al., "Internet of Things: Architectural Framework for eHealth Security," Journal of ICT Standardization, vol. 1, no. 3, pp. 301–328, January 2014. doi:10.13052/jicts2245-800X.133.
9. G. Suciu et al., "Smart Cities Built on Resilient Cloud Computing and Secure Internet of Things," in 2013 19th International Conference on Control Systems and Computer Science, May 2013, pp. 513–518. doi:10.1109/CSCS.2013.58.
10. P. Kamal et al., "Evaluating the Efficiency and Effectiveness of a Federated SSO Environment Using Shibboleth," Journal of Information Security, vol. 6, no. 3, pp. 166–178, July 2015.
11. D. Puthal et al., "A Dynamic Key Length Based Approach for Real-Time Security Verification of Big Sensing Data Stream," 2015, pp. 93–108. doi:10.1007/978-3-319-26187-4_7.
12. F. Roesner et al., "World-Driven Access Control for Continuous Sensing," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 1169–1181. doi:10.1145/2660267.2660319.
13. L. Belli et al., "Design and Deployment of an IoT Application-Oriented Testbed," Computer, vol. 48, no. 9, pp. 32–40, September 2015. doi:10.1109/MC.2015.253.
14. M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," in 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), June 2012, pp. 423–430. doi:10.1109/CLOUD.2012.103.
15. B. Vieira and E. Poll, "A Security Protocol for Information-centric Networking in Smart Grids," in 1st ACM Workshop on Smart Energy Grid Security (SEGS'13), November 2013. doi:10.1145/2516930.2516932.

ABOUT THE AUTHORS

Luca Davoli is a postdoctoral research associate at the Department of Engineering and Architecture of the University of Parma, Italy. He received his Dr. Ing. (Laurea) degree in Computer Science from the University of Parma, Italy, in 2013. In 2017, he received his Ph.D. in Information Technologies at the Department of Information Engineering of the same university. His research interests are Internet of Things, P2P Networks, Pervasive Computing, Big Stream and Software-Defined Networking. He is an IEEE Graduate Student Member. Contact him at luca.davoli@unipr.it.

Laura Belli is a postdoctoral research associate at the Department of Engineering and Architecture of the University of Parma, Italy. She received her Dr. Ing. (Laurea) degree in Computer Science from the University of Parma, Italy, in 2011. In 2016, she received her Ph.D. in Information Technologies at the Department of Information Engineering of the same university. Her research interests are Internet of Things, Pervasive Computing, Big Stream, Database Integration, Mobile Computing. Contact her at laura.belli@unipr.it.

Luca Veltri is an assistant professor at the Department of Engineering and Architecture of the University of Parma, Italy, and he teaches classes on Communication Networks, and Network Security. He is also the director of the UniPR Co-Lab, a cross-department research center in educational technology at University of Parma. From 1999 to 2002, before joining the University of Parma, he has been with CoRiTeL, a research consortium founded by Ericsson Telecomunicazioni, where he leaded different research projects in networking and multimedia communications. He participated also in several research projects founded by the European Union, by the European Space Agency, and by the Italian Ministry of University and Research. His current research interests include Internet of Things, Software-Defined Networking, and Network Security. He is co-author of more than 70 papers on international conferences and journals. He is an IEEE Member. Contact him at luca.veltri@unipr.it.

Gianluigi Ferrari is an associate professor of Telecommunications at the Department of Engineering and Architecture of the University of Parma, Italy, where he coordinates the Internet of Things Lab (http://iotlab.unipr.it). He has published extensively in the areas of advanced communication and networking, signal processing, IoT and smart systems, receiving paper/technical awards at IWWAN 2006, EMERGING 2010, BSN 2011, ITST 2011, SENSORNETS 2012, EvoCOMNET 2013, BSN 2014, SoftCOM 2014, EvoCOMNET 2015. He currently serves on the editorial boards of several international journals. He is an IEEE Senior Member. Contact him at gianluigi.ferrari@unipr.it.

Consistent Disaster Recovery for Microservices: the BAC Theorem

Guy Pardon
Atomikos

Cesare Pautasso
Università della Svizzera Italiana, Lugano, Switzerland

Olaf Zimmermann
Hochschule für Technik Rapperswil (HSR FHO), Switzerland

How do you back up a microservice? You dump its database. But how do you back up an entire application decomposed into microservices? In this article, we discuss the tradeoff between the availability and consistency of a microservice-based architecture when a backup of the entire application is being performed. We demonstrate that service designers have to select two out of three qualities: backup, availability, and/or consistency (BAC). Service designers must also consider how to deal with consequences such as broken links, orphan state, and missing state.

Microservices are about the design of fine-grained services, which can be developed and operated by independent teams, ensuring that an architecture can organically grow and rapidly evolve.[1] By definition, each microservice is independently deployable and scalable; each stateful one relies on its own polyglot persistent storage mechanism. Integration at the database layer is not recommended, because it introduces coupling between the data representation internally used by multiple microservices. Instead, microservices should interact only through well-defined APIs, which—following the REST architectural style[2]—provide a clear mechanism for managing the state of the resources exposed by each microservice. Relationships between related entities are implemented using hypermedia,[3] so that representations retrieved from one microservice API can include links to other entities found on other microservice APIs. While there is no guarantee that a link retrieved from one microservice will point to a valid URL served by another, a basic notion of consistency can be introduced for the microservice-based application, requiring that such references can always be resolved, thus avoiding broken links. As the scale of the system grows, such a guarantee can be gradually weakened, as is currently the case for the World Wide Web.

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

Microservice architectures are designed to survive individual microservice failures and to prevent cascading failures of all services.[4] Backing up and recovering a microservice is an important operation, which allows it to survive significant failures affecting its operating environment.[5] For example, if one node of a cluster crashes, it should be possible to restart the microservice on another node and connect it to the same database so that it can access its corresponding state. If the database server itself crashes, it is possible to recover the database from previously executed dumps that are stored elsewhere. Backups can be performed from within the database or using a backup data pump, which periodically and incrementally synchronizes replicated databases.

This article is concerned with the problem of backing up an entire microservice architecture where a running application decomposed into multiple microservices needs to be backed up so that it can be recovered after a disaster strikes. How should one do so and still guarantee a certain level of consistency between the various microservices? How should one do so without affecting the availability of the application? In particular, we are concerned with the full availability of the application, during which it is possible to both read its state and perform updates to any of its microservices. Depending on the chosen type of backup technique, it might be necessary to suspend normal operation and only allow for performing read operations, to ensure that the underlying state of each microservice is being backed up consistently.

EVENTUAL INCONSISTENCY

Following a domain-driven design methodology,[6] a monolithic application (see the left image in Figure 1) may be split into different Bounded Contexts. This practice is beneficial to ensure the internal semantic consistency of each domain context, as well as to enable the interoperability of data exchanged between separate domains across a Context Map. This method is also recommended when splitting up the monolith into multiple microservices (see the right image in Figure 1) that remain logically connected by a set of loose relationships between their data models. For example, a sales-related application is split into one microservice to handle customers, one to manage the product catalog and warehouse inventory, another to manage orders, and yet another to handle the shipping process. Clearly, a shipment must refer to a given customer and to a specific order, which should refer to the actual products (see Figure 2).

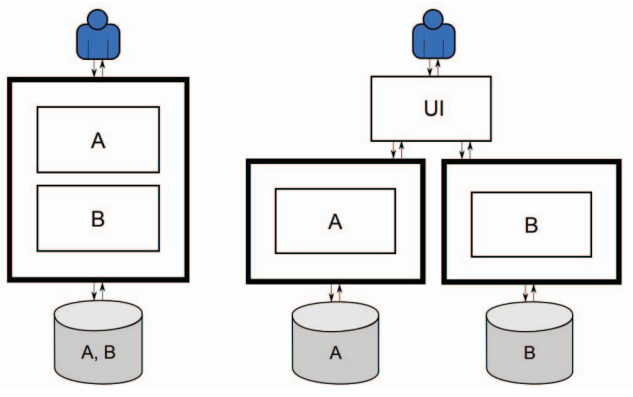


Figure 1. (Left) monolithic architecture and (right) microservice architecture.

While normal operations will lead to the eventual consistency[7–8] of the state of the application partitioned across multiple microservices, when disaster strikes, the state of the recovered application will eventually become inconsistent due to snapshot timing issues.

Even if every backup of individual microservices can be trusted for their independent recovery, as a whole, it is likely that the state of the application will not converge and will remain inconsistent when each microservice has been backed up and recovered independently. For example, it might be problematic if after the independent recovery of each microservice some orders would refer to missing product descriptions or some shipments would point to customers no longer present in the corresponding recovered microservice.

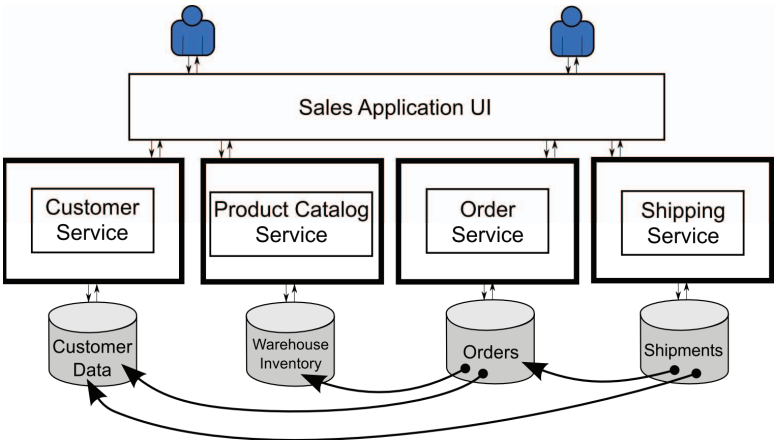


Figure 2. An example microservice architecture with references spanning across individual microservices.

To avoid this, it is important that the snapshot taken to back up every microservice is taken not necessarily at the same time, but when the overall application state was consistent. While the backup is underway, clients may not be allowed to perform arbitrary update operations spanning multiple microservices. Depending on the size of the snapshot and whether a full backup or an incremental backup technique is performed, this might require a non-negligible amount of time.

Alternatively, if introducing extra data loss during recovery is not a problem, it would be possible to achieve consistency across all microservices by rolling back the state of every microservice to the point in time when the oldest backup of one microservice was taken. This is not acceptable in many application scenarios and contexts.

THE BAC THEOREM

In this article, we present the BAC theorem for microservices, which states:

> *When backing up an entire microservices architecture, it is not possible to have both availability and consistency.*

The theorem is inspired by the CAP theorem,[9] trading off consistency against availability in distributed (replicated) databases. The informal proof is similar. If we allow each microservice to evolve its state independently and perform a backup of every microservice at a different time, the whole set of backups will not offer a consistent view over the state of the whole application. In case of recovery, each of its microservices will be restarted from a snapshot taken independently, and, thus, it is possible that referenced entities between microservices will be missing. As opposed to normal operation when this is allowed to happen because the various microservices will eventually reach a consistent state,[10] in case of recovery, this will not happen because the clients performing distributed transactions[11] over the microservices will be long gone.

The alternative to ensure consistency would be to lock the entire application during the backup and make sure that the state of its microservices is snapshotted at the same time. Locking the entire application would compromise its full availability, because only read operations would be allowed while backing up. Depending on the duration of the backup (which is bound by the slowest microservice), this might affect the clients whose requests to change the state of some microservice might have to be denied or delayed until the backup of the entire application is completed.

Conceptual Model: Commands and Events

Our abstract model of a microservice is inspired by domain-driven design (DDD),[6] which introduces patterns such as Service, Aggregate, and Domain Event to decompose the business logic of an application into discrete, loosely coupled processing units, each of which has a single reason to change.[12]

A microservice accepts "commands"—either through user input or by processing "domain events" from other microservices. Commands can fail due to validation or integrity constraints by the receiving service. When processed successfully, commands can result in the publication of one or more events reflecting the outcome of the command. Events do not "fail" because they are merely notifications of things that already happened. Events can be lost, though, in particular (for this article) due to recovery from an obsolete backup. A microservice that receives an event can convert it into a command for local processing. Such a command in turn can fail (in particular if the microservice does not understand the event after disaster recovery).

Microservices refer to each other's data through loosely coupled references. If microservices expose a RESTful HTTP API, these references naturally correspond to resource identifiers. Each data item is owned by exactly one microservice and referenced by any number of other microservices.

Without loss of generality, microservices are using the event storage pattern[13] for managing changes applied to their state. The database of the microservice thus contains a log of business events, which can be replayed to bring the state of the microservice to its latest version, while keeping track of its history for auditing and debugging purposes.

Some of the events in the log are local (such as the creation of a new customer record). Some events conceptually cross the boundary between the two microservices. For example, when a customer orders a new shipment, an event will be added to the shipment microservice (the shipment order) and another will be added to the customer microservice (the reference to the shipment order in the customer shipment collection). More generally, in this case, there is an interaction between microservices (the request to create a new shipment order for a given customer), which results in a state transition for both of them. Thus, a new event is added to the log of each microservice.

Eventual Inconsistency with Full Availability

When making an independent backup of the state (the event log) of each service, it is possible that one service will be backed up before the event is added to the log, while the other service will be backed up after the corresponding event is added to its log. When restoring the services from backup, only one of the two services will recover its complete log. The global state of the microservice architecture will thus become inconsistent after restoring it from backup. In a similar way, during the execution of a particular use case, it is possible that one service is backed up before a command is executed in it, while another service is backed up after the corresponding command (of the same use case) is executed in it.

In a hypermedia context, the inconsistency typically manifests itself after the recovery from backup of the services in the following ways:

- *Broken link*. This is when the reference can no longer be followed—for example, where, using database terminology, foreign key records are recovered without the corresponding primary key records. In Figure 3, the up-to-date Microservice A refers to an obsolete version of the other Microservice B, where the referenced entity cannot be found after it has been recovered. The inconsistency can thus be detected when clients try to follow the reference from A to B and instead find that the link is broken.
- *Orphan state*. This is when there is no reference to be followed—for example, where primary key records are recovered without the corresponding foreign key records. In Figure 4, the state of the up-to-date Microservice A is no longer referenced from the state of the Microservice B recovered from an obsolete backup. This situation might be

more difficult to detect from clients, because there are no immediately visible inconsistencies when they interact with either service. However, this might introduce storage space leaks for A, if there is no garbage-collection mechanism for such orphan states. Moreover, duplicate event log entries for A might be introduced (which might lead to unwanted side effects), as the obsolete Microservice B is brought up to date.

In the general case, we call this phenomenon "*missing state*"—an example of which is shown in Figure 5: Microservice B remains consistent with A until it crashes. After recovery from an obsolete backup, B does not have the state corresponding to the latest events logged by A.

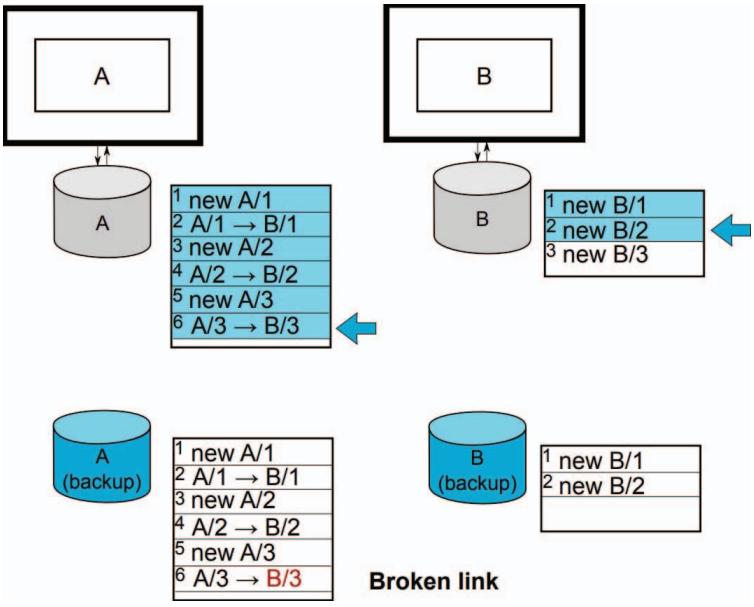


Figure 3. Broken link. The event logs tracking the state updates of each microservice are backed up independently at different times. The blue arrows indicate the points in time in which the backup snapshot of each log was taken. The blue events are the ones that have been copied to the backup. A/3 → B/3 shows that there is a reference from A to B. When the state of Microservice B is recovered, it will be inconsistent with respect to Microservice A referring to it. However, after recovering from the backup, B/3 is no longer part of the recovered state of service B, hence the broken link.

Consistent Backups with Limited Availability

It becomes possible to avoid the inconsistency by coordinating the backup of the two services to ensure that a snapshot is taken either before the interaction takes place or after the effects of the interaction have been logged on both sides. This means that, while the backup is running, no events can be added to the microservice logs, effectively reducing the availability of the microservices for clients that need to perform some state transitions/write operations.

This, however, violates the independence of the two microservice disaster recovery processes, as it introduces tight coupling into their operational lifecycles. While this might be possible to achieve for a small number of microservices, chances are that it will become impractical as the number of related microservices grows. The mere number of microservices is not problematic as such, but things get difficult if many or all of them have relationships with each other. The semantic complexity of the domain and the chosen microservice decomposition granularity will avoid or exacerbate the problem.

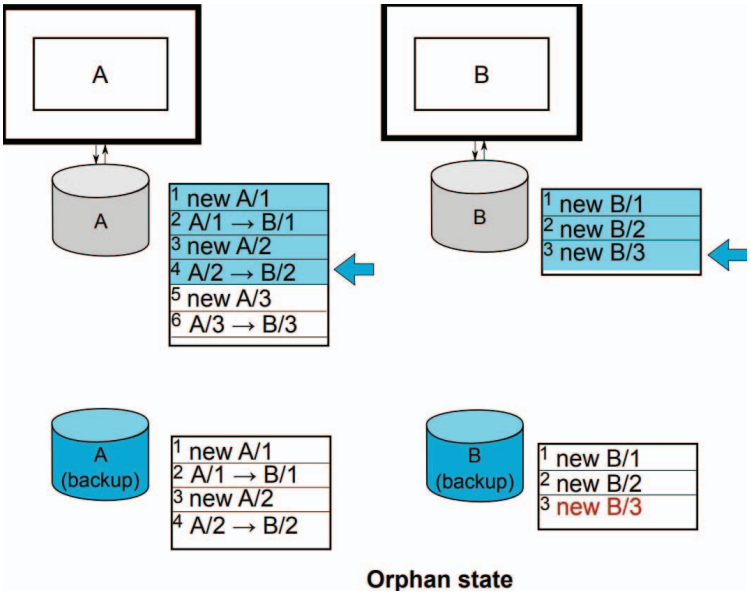


Figure 4. Orphan state. The event logs are backed up independently at different times, indicated by the blue coloring. When the state of Microservice B is recovered, it will be inconsistent with respect to Microservice A, which is no longer referring to it. In the figure, the reference A/3 → B/3 is not backed up. So, after recovery, B/3 is orphaned, as there is no longer a reference to it from A.

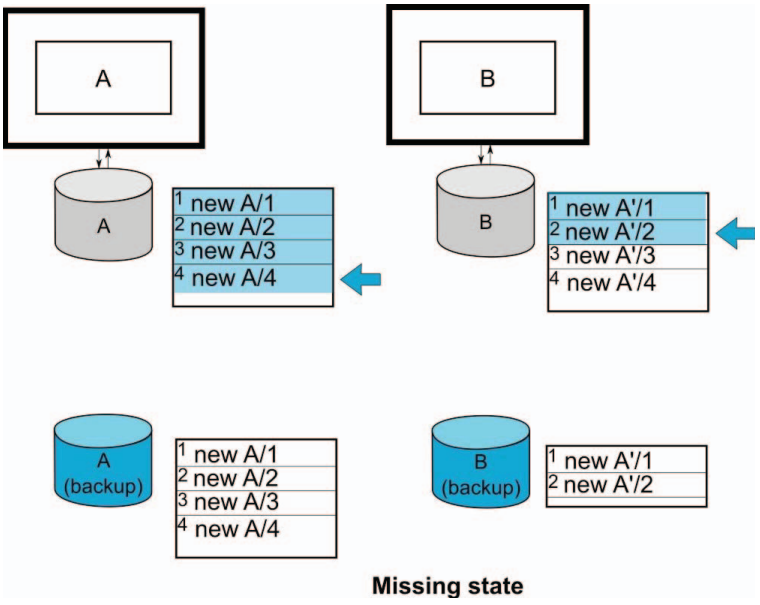


Figure 5. Missing state. A and A' are service-specific representations of related entities affected by the same events. The event logs are backed up independently at different times (blue coloring). When the state of Microservice B is recovered, it will be inconsistent with respect to Microservice A, as the entities A'/3, A'/4 corresponding to A/3, A/4 are missing.

DEALING WITH THE CONSEQUENCES OF THE BAC THEOREM

How should one design microservices that are aware of the consequences of the BAC theorem and can thus deal with the possible eventual inconsistency introduced by their independent backup and recovery?

Dealing with Broken Links

Broken links in A after a restore (with old data) of B are equivalent to long-duration unavailability of B during normal operation. Typical strategies include:

1. *Reconstructing B to contain a valid reference*. A human (possibly the user) could be asked to restore the state of B to a correct one, including the missing reference. Until reconstruction happens, A will have to tolerate the missing reference (which it has to do during normal operations anyway, when temporary network glitches cause the unavailability of B). In some cases, reconstruction of B might not be possible. For instance, if B was a reservation of a scarce resource (like an airplane seat on a flight), the flight might be fully booked in the meantime. Restoring a reservation system with data loss probably results in overbooking, a common source of headaches for most airlines.
2. *Doing nothing*. The system and its users can accept that some parts of the system can be inconsistent. This strategy can be acceptable if the reference to B is only there for informational purposes (not really needed for A to function). As the scale of the application grows larger with an increasingly large number of microservices, users might not expect full consistency at all times. Business criticality and quality attributes such as accuracy and auditability, as well as related compliance controls, are key decision drivers here.
3. *Caching*. Microservice A could cache whatever data it needs from B, so that it has at least something to work with. Caching can work through either batch exports and imports, storing events from B in a cache at A, or fetching all relevant data from B when A establishes the reference.

When processing user commands, Strategies 1 and 3 seem like reasonable options: Either the user rectifies any problems, or the system uses whatever it remembers in its cache to proceed. But what about processing commands that result from incoming events? In other words, what does a service do if it receives events and detects broken links? In that case, Strategy 1 is not very practical because the user is typically not involved in event processing. Instead, the system could use Strategy 3 to have the missing data cached, or it could process with warnings and resort to Strategy 1 at a later time when a user logs in. Lastly, the system could ignore any missing data, using Strategy 2. In any case, broken links should be logged as they are detected so that operators can decide which strategy to follow when attempting to restore the consistency of the application during disaster recovery.

Dealing with Orphan State

Orphan state may or may not be problematic, depending on the application. Some form of human intervention is required to decide what to do with orphan state.

When processing user commands, orphans can be detected, as some microservices might already store information provided by the user. Thus, a solution could be to have the user interface or an API Gateway assume responsibility for flagging orphan candidates so that the users of the system can easily fix any anomalies. It is recommended to regularly run a validation service and/or database consistency checker.

What about processing commands that result from incoming events? Here, orphan state might cause validation errors and, consequently, the rejection of the incoming event. This is probably not desirable because orphan state should not disturb the normal system operation after a restore from backup. A lot of events can come in in high-volume systems—rejecting incoming events could cause a spike of errors or warnings that will be difficult to deal with in practice. A safer

strategy would be to treat incoming events as the latest version of the truth (which, in a way, they are) and ignore validation errors when processing incoming events. The careful developer will place a warning in the logs when this happens.

Dealing with Missing State

If events are easy to reproduce, B might simply ask for a replay of missed events. This requires some authoritative source where B can ask.

If events are not easy to reproduce from their source, hopefully the event delivery infrastructure (if any) offers some form of reliability. If not, manual intervention in B might be needed to (re)apply the missing state in the form of one or more user commands.

In all other cases, the only option might very well be to accept that events can be lost.

DISCUSSION

Avoiding the Problem

When the challenges posed by BAC are too hard to solve, avoiding the problem can be your only option: Make sure your backups are consistent. If you want to benefit from "referential integrity" (in the loose sense of the word), place related data in the same database (possibly using separate schemas) so they are backed up together. Hence, your microservices should be merged together as far as their state management is concerned (see Figure 6).

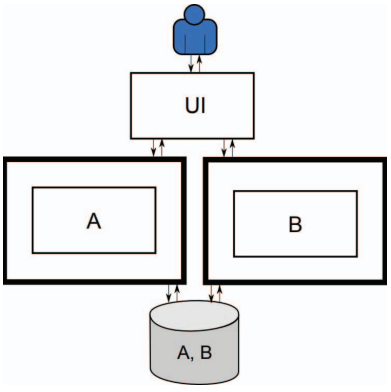


Figure 6. Microservice architecture with shared database, trivially enabling consistent recovery but introducing undesired coupling between the microservices.

Acknowledging the Problem

An alternative strategy might simply be to be aware of the problem and the resulting data losses or eventual inconsistencies. For some applications, this can be an acceptable option. This strategy must be carefully chosen, documented, and approved; at runtime, suited log entries should be created, and warnings should be sent to system administrators if justified in the given application scenario.

Let the Business Guide You

Microservices can be modelled around the existing departments and value streams of the business. Each business department is responsible for its own data. The proven communication between and autonomy of these departments can hint at how to ensure that your microservices are

sufficiently decoupled to work in practice.[12] In addition, the anomalies discussed in this article are probably known (indirectly) by the business already; chances are that appropriate policies exist already to take care of them.

The Architect's Job

We've discussed a few options for dealing with BAC. There is no one-size-fits-all solution. The architect's job is to decide which options work best for which set of microservices, especially in light of introducing an appropriate disaster recovery plan, dealing with how to perform backups (independently for individual microservices or globally for the entire system), and dealing with the consequences of restoring a system from potentially inconsistent backups (broken links, orphan state, and missing state). Thinking about these concerns before actually building your microservice ecosystem can save a lot of headaches afterwards.

RELATED WORK

High Availability and Replication

When microservices rely on a highly available, replicated database,[14] or when they are built as a replicated state machine,[15] one could argue that there is no need for an additional backup-and-recovery solution. Because there should always be $N > 1$ replicas of the state (possibly stored in databases distributed across different locations, availability zones, or datacenters), the backup process is happening continuously, and the recovery can theoretically be instantaneous.

Introducing a high-availability data layer for every microservice is, however, an expensive solution that greatly increases the operational complexity of the system. Likewise, even if microservices design principles married with DevOps and continuous integration practices appear to hint towards a world made of unstoppable systems whose services are in permanent operation, the question is whether it is wise to bet there will never be a need for recovering such systems from scratch. In general, the steep investment into a dedicated, highly available database solution with full replication to support every microservice might turn out to be difficult to justify. This ultimately needs to be compared against the cost of dealing with the eventual inconsistency of the application after its recovery.

Distributed Snapshots

Distributed snapshot algorithms[16] are also applicable in our scenario for the global state detection and check-pointing of the entire architecture. Without the need for sharing a common clock or accessing shared memory, it is possible to coordinate the various microservices as they store their state into a snapshot for backup purposes. Additionally, the distributed snapshot algorithm is not supposed to interfere with the ongoing distributed computation, which would fit with the need to preserve the full availability of the microservices.

However, the assumption of message-based, asynchronous communication over error-free, order-preserving channels whose content can also be snapshotted only partially fits with existing microservice architectures, where interactions are implemented using both message queues and the synchronous HTTP protocol. Likewise, ensuring that all microservices are designed to comply with Lamport and Chandy's requirements[16] would strongly couple all microservices and limit the flexibility in their independent deployment, operation, and evolution.

Interaction Contracts

Going beyond traditional database recovery, Barga, Lomet, and Weikum proposed a comprehensive form of recovery for multi-tier applications with communicating components.[17] The approach was based on interaction contracts between persistent components assumed to behave piecewise deterministically. This way, it is possible to recover the state of a component, starting

from a known initial state (such as when the component was deployed) by replaying every message it received in the same order they reached the component before the crash. This approach is, hence, also based on the notion of restoring a consistent state by replaying messages that would carry some form of commands and/or events in our frame of reference.

Other Service Design Issues and Coupling Criteria

Many architectural decisions and other hard design problems in service-oriented systems implemented with microservices are identified and partially answered in C. Pautasso et al.[12] How to handle backups remains an open problem according to the literature on microservice design in industry and academia.

Loose coupling has many dimensions, including time, location, platform, and format.[18] The Service Cutter methods and tool suggests microservices decompositions and re-compositions based on 16 functional and non-functional coupling criteria.[19] These criteria can help decide for one or more of the strategies for dealing with the consequences of the BAC theorem that we presented in this article. Backup requirements and the BAC theorem can also be seen as an additional coupling criterion.

CONCLUSION

While splitting the monolith is the mantra of microservice architectures, it is important to realize that the granularity of the result depends on what you are trying to achieve. Increased development velocity can be obtained with many multiple loosely coupled development teams, which can deploy new versions of their microservice at will. However, the long-term sustainability of the application might be harmed if, in case of disaster, it will be impossible to achieve a holistic recovery that brings back all microservices in a globally consistent state. Thus, it might be necessary to cluster together multiple microservices, which should be backed up in lockstep (see Figure 6). While this might not be necessary for all microservices into which the monolith has been decomposed, chances are that the granularity of the microservices to be consistently backed-up will be larger.

REFERENCES

1. S. Newman, *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, 2015.
2. L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*, O'Reilly Media, 2013.
3. M. Amundsen, *Building Hypermedia APIs with HTML5 and Node*, O'Reilly Media, 2011.
4. T. Killalea, "The hidden dividends of microservices," *Communications of the ACM*, vol. 59, no. 8, 2016, pp. 42–45; https://dl.acm.org/citation.cfm?doid=2948985.
5. L. Bass, I. Weber, and L. Zhu, *DevOps:A Software Architect's Perspective*, Addison-Wesley, 2015.
6. E. Evans, *Domain-driven design: tackling complexity in the heart of software*, Addison-Wesley, 2004.
7. W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, 2009, pp. 40–44; https://dl.acm.org/citation.cfm?id=1435432.
8. D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior," *Proc. of the 6th Workshop on Middleware for Service Oriented Computing* (MW4SOC), 2011, p. 1:1; https://dl.acm.org/citation.cfm?doid=2093185.2093186.
9. E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, 2012, pp. 23–29; http://ieeexplore.ieee.org/document/6133253/.
10. P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Communications of the ACM*, vol. 56, no. 5, 2013, pp. 55–63; https://dl.acm.org/citation.cfm?doid=2447976.2447992.

11. G. Pardon and C. Pautasso, "Atomic distributed transactions: A RESTful design," *Proc. of the 5th International Workshop on Web APIs and RESTful Design* (WS-REST 2014), 2014.
12. C. Pautasso et al., "Microservices in practice, part 1: Reality check and service design," *IEEE Software*, vol. 34, no. 1, 2017, pp. 91–98; http://ieeexplore.ieee.org/document/7819415/.
13. M. Fowler, "Event Sourcing," *martinfowler.com*, 2005; https://martinfowler.com/eaaDev/EventSourcing.html.
14. B. Kemme and G. Alonso, "Database Replication: A tale of research across communities," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, 2010, pp. 5–12.
15. F.B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, 1990, pp. 299–319; https://dl.acm.org/citation.cfm?id=98167.
16. K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, 1985, pp. 63–75; https://dl.acm.org/citation.cfm?id=214456.
17. R.S. Barga, D.B. Lomet, and G. Weikum, "Recovery guarantees for general multi-tier applications," *Proceedings of the 18th International Conference on Data Engineering* (ICDE2002), 2002, pp. 543–554.
18. C. Pautasso and E. Wilde, "Why is the web loosely coupled? A multi-faceted metric for service design," *Proc. of the 18th World Wide Web Conference* ((WWW2009)), 2009, pp. 911–920.
19. M. Gysel et al., "Service Cutter: A systematic approach to service decomposition," *Service-Oriented and Cloud Computing - 5th IFIP WG 2.14 European Conference* (ESOCC), 2016, pp. 185–200.

ABOUT THE AUTHORS

Guy Pardon is the chief architect at Atomikos, where he leads the development of both traditional and modern (service-oriented) transaction technology. He occasionally speaks at conferences, publishes technical articles, and teaches trainings related to mission-critical enterprise application development—his main professional interest. His main research interest concerns reliability for distributed systems. Pardon has a PhD from ETH Zurich, Switzerland. Contact him at guy@atomikos.com.

Cesare Pautasso is a professor at the Faculty of Informatics, Università della Svizzera Italiana (USI) Lugano, Switzerland, where he leads the Architecture, Design, and Web Information Systems Engineering research group. He supervises the research of five PhD students who are building experimental systems to explore the intersection of cloud computing, software architecture, Web engineering, and business-process management. Previously, he was a researcher at the IBM Zurich Research Lab and a senior researcher at ETH Zurich, from which he has a PhD. He is the coauthor of the books "SOA with REST" and "Just Send an Email: Anti-patterns for email-centric organizations," and he is the co-editor of the *IEEE Software* Insights department. Find more information at www.pautasso.info. Follow him @pautasso. Contact him at c.pautasso@ieee.org.

Olaf Zimmermann is a professor of software architecture and an institute partner at the Institute for Software at the Hochschule für Technik Rapperswil (HSR). His research areas include service-oriented computing and architectural knowledge management. Previously, he was a senior principal scientist at ABB Corporate Research and a researcher and executive IT architect at IBM. He has a PhD from Stuttgart University. The Open Group awarded him a Distinguished IT Architect (Chief/Lead) Certification. He is a book author and a co-editor of the Insights column in *IEEE Software*. Contact him at olaf.zimmermann@hsr.ch.

SECURE: Self-Protection Approach in Cloud Resource Management

Sukhpal Singh Gill
The University of Melbourne, Australia

Rajkumar Buyya
The University of Melbourne, Australia

In the current scenario of cloud computing, heterogeneous resources are located in various geographical locations requiring security-aware resource management to handle security threats. However, existing techniques are unable to protect systems from security attacks. To provide a secure cloud service, a security-based resource management technique is required that manages cloud resources automatically and delivers secure cloud services. In this paper, we propose a self-protection approach in cloud resource management called SECURE, which offers self-protection against security attacks and ensures continued availability of services to authorized users. The performance of SECURE has been evaluated using SNORT. The experimental results demonstrate that SECURE performs effectively in terms of both the intrusion detection rate and false positive rate. Further, the impact of security on quality of service (QoS) has been analyzed.

Quality of service (QoS) plays an important role in the era of cloud computing in which delivered cloud services are measured and monitored in terms of QoS to ensure their availability. Nevertheless, offering committed cloud services that guarantee customer's changing QoS needs while precluding them from security attacks is a big challenge.[1] Provisioning and scheduling cloud resources is often done based on their availability without providing the required security.[2] To make cloud computing systems more effective, the security requirements of every cloud component should be satisfied. To realize this, a security-based resource allocation mechanism is required that automatically manages cloud resources and delivers secure cloud services.

Self-protection is the ability of a computing system to defend itself against threats and intrusions. A self-protection component aids in distinguishing and recognizing intimidating behavior and reacts autonomously to protect itself against malicious attacks.[3] These systems defend themselves from attackers by differentiating illegitimate from legitimate behavior and performing the

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

required actions to block such attacks without user awareness. Table 1 shows the list of security attacks, from which a system must be self-protected.[2-10]

Table 1. List of Security Attacks

Classification of Attack	Description	Attack Name
Denial of Service (DoS)	Attacker generates a large amount of network traffic, which damages the victim's network (in terms of QoS) by flooding.	SMURF: ICMP (Internet Control Message Protocol) Used to create DoS, in which a pointing packet generates echo requests toward the broadcast IP address. LAND (Local Area Network Denial): Attacker transfers spoofed SYN packet in a TCP/IP network when the destination and source address are the same. SYN Flood: To reduce storage efficiency, an attacker sends IP-spoofed packets to crash the system. Teardrop: Exploits a flaw in the deployments of older TCP/IP stacks.
Distributed-DoS (DDoS)	A DDoS attack occurs when several systems flood the bandwidth or resources of a victim's system, generally one or more Web servers.	HTTP Flood: Attacker exploits seemingly legitimate HTTP GET or POST requests. Zero Day Attack: A security loophole in a cloud based system that is unknown to the developer or vendor.
Remote to Local (R2L)	Attacker executes commands to get access to the system by compromising the network (in terms of QoS).	SPY: Installs itself secretly on a system and runs in the background for phishing. Password Guessing: Attackers guess passwords locally or remotely. IMAP (Internet Message Access Protocol): Finds an IMAP Mail server which is known to be vulnerable.
User to Root (U2R)	To destroy the network, attacker gets root access into the system.	Rootkits: Offers constant privileged access to a system while actively hiding its existence. Buffer Overflow: Occurs when a program copies a large amount of data into a static buffer.
Probing	To breach the personal information of victim, an attacker uses different programming languages.	Ports Sweep: Multiple hosts are scanned for a particular listening port. NMAP (Network MAPper): Performs port scanning.

Recently, researchers focused on identifying new techniques for detection and prevention of intrusions in computing systems and discovered that the Intrusion Detection System (IDS) is an effective way to protect network from attacks. IDS stops attacks, performs recovery after attacks, and investigate security loopholes to help avoid such problems in the future. IDS can be categorized into two types based on anomaly and signature. Signature-based IDS is used to detect the signatures of known attacks in the database, while anomaly-based IDS analyzes abnormal activities. SNORT[13] is the most effective IDS that can be used for attack detection. Different machine learning techniques are used for anomaly-based IDS, but State Vector Machine (SVM) is the most commonly used anomaly-based detector.[2,3]

This article proposes a **S**elf-prot**E**ction approa**C**h in clo**U**d **R**esource manag**E**ment (SECURE) approach for dealing with security attacks. SECURE can create new signatures automatically and provide security against DDoS, Probing, U2R, R2L and DoS security attacks. Based on MAPE-K loop, an algorithm for different phases has been developed to monitor, analyze, plan and execute. SECURE continuously monitors security attacks during the execution of workloads, performs analysis to understand alerts in the case of security attacks, makes a plan to perform required actions to manage threats, and executes the action. Security agents (sensors) are created on SVM as an anomaly detector. SECURE increases the security of cloud-based services and increases intrusion detection rate if the same threat arrives again.

SELF-PROTECTION IN CLOUD: STATE-OF-THE-ART

A Secure Autonomic Technique (SAT) was proposed to provide a secure cloud environment for execution of user applications.[4] SAT detects DoS (SYN Flood) attacks with excellent data accuracy while managing cloud services. Carpen-Amarie[5] proposed the Self-Adaptive Data Management (SADM) system, which manages large amounts of data through the Nimbus cloud environment. SADM also integrates a security policy as part of its framework to detect DoS (Teardrop) security attacks to discover malicious clients. Wailly and colleagues introduced a Virtual Environment based Self-Protecting Architecture (VESPA)[6] to protect cloud infrastructure resources using autonomic security loops. VESPA's performance is measured in terms of response time while detecting U2R (rootkit) security attacks. Sulistio and Reich[7] proposed a Self-Protecting Cloud Service (CPCS) architecture to reduce the barrier for cloud adoption, which decreases Service Level Agreement (SLA) violations. Cloud services of various providers (e.g. Microsoft Azure, Amazon EC2 and Google App Engine) are compared based on the R2L (SPY) security attack for different infrastructure configurations.

To protect a network against malware such as DoS (LAND) security attacks, Benkhelifa and Welsh[8] proposed a mechanism called Malware Inspired Cloud Self-Protection (MICSP), which uses signature analysis to detect malware and prevent the system from future such attacks. Paul[9] proposed a Cloud based Trust Management System (CTMS) to address authentication, authorization and data integrity of cloud security; it also measures the effect of DoS security attacks on the latency introduced during the processing of jobs. Di Pietro and colleagues proposed a Secure Management technique for Virtualized Resources (SMVR),[10] which detects U2R (buffer overflow) security attacks at runtime to improve virtualization security. SMVR reduces security breaching and improves memory utilization. Sarhan and Carr proposed a Self-Protection Data Scheme (SPDS),[11] which uses active data bundles and agent-based secure multi-party computation to protect sensitive data outsources to a cloud for processing. It further provides a secure key management using the RSA algorithm to protect form R2L (IMAP) security attacks.

Table 2 shows a critical comparison of SECURE with existing approaches based on different criteria. The existing resource management techniques considered only one type of the security attacks from DoS, R2L, U2R but all three types of security attacks have not been considered at the same time to the best of our knowledge. Moreover, there is a need to prevent DDoS and Probing security attacks.

Table 2. Comparison of SECURE with Existing Approaches

Year	Technique	Autonomic Mechanism	Type of Attacks	Analyzed Impact of Security on QoS	Performance Parameters
2010	SAT [4]	✓	DoS	✕	Resource Utilization
2011	SADM [5]	✓	DoS	✕	Throughput
2012	VESPA [6]	✓	U2R	✕	Response Time
2013	CPCS [7]	✓	R2L	✕	SLA Violations
2014	MICSP [8]	✓	DoS	✕	Resource Utilization
2015	CTMS [9]	✕	DoS	✕	Latency
2016	SMVR [10]	✕	U2R	✕	Resource Utilization
2017	SPDS [11]	✓	R2L	✕	Resource Utilization
2018	SECURE (Proposed)	✓	Probing, DoS, R2L, U2R and DDoS	✓	Intrusion Detection Rate, Response Time, Signature Generation Rate and False Positive Rate

SECURE protects a cloud-based computing system from five different types of security attacks including DDoS, Probing, U2R, R2L and DoS and analyzes the impact of security on QoS. Further, the performance of SECURE has been tested in terms of response time, false positive rate, and intrusion detection rate. Performance evaluation shows that SECURE performs effectively.

SECURE: SELF-PROTECTION APPROACH IN CLOUD RESOURCE MANAGEMENT

This section presents the architecture of SECURE, which offers self-protection against security attacks. Figure 1 shows the architecture of SECURE and includes the following sub-units:

- Cloud User: Cloud users submit their requests for execution.
- Request Service Handler: A buffer to store information about every request and pass the different workloads to the Dispatcher Service.
- QoS Manager: Identifies the different QoS requirements of a user request and forwards it to the Dispatcher Service.
- Dispatcher Service: Dispatches the user workloads along with their QoS requirements to Detection Engine.
- Detection Engine: Uses two types of IDS, which uses SNORT search for signatures of known attacks in the database (SNORT DB) and uses an SVM-based anomaly detector to analyze abnormal activities (unknown attacks). The training dataset is used to design SVM to find and diagnose input network traffic data to identify the attack. An action is taken once an attack is detected and stored into the database.
- Resource Provisioner: Resources are provisioned using Q-aware,[3] in which suitable resources are identified for a particular workload based on their QoS requirements as designated by QoS Manager. These resources are then provisioned as per the user requests.
- Resource Scheduler: QRSF[12] is used to schedule the provisioned resources with minimum execution cost and time.

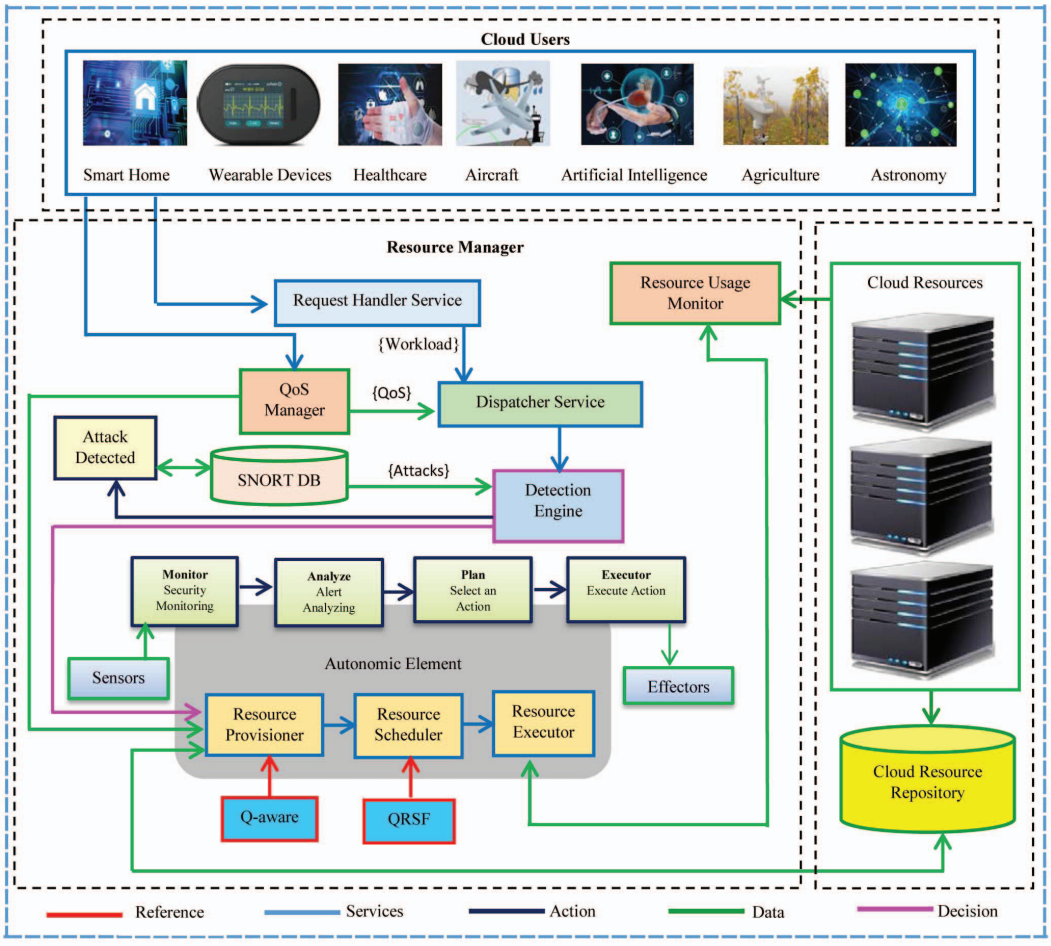


Figure 1. SECURE Architecture.

- Resource Executor: Executes the workloads using the scheduled resources.
- Autonomic Element: Comprises six components: sensor, monitor, analyze, plan, executor, and effector.
- Resource Usage Monitor: Measures the value of resource utilization during workload execution.
- Cloud Resource Repository: Stores the configuration of the cloud resources.

SECURE defends itself from attackers by differentiating illegitimate from legitimate behavior and performing the required actions to block those threats without user awareness. The threats covered by SECURE are DDoS, Probing, U2R, R2L, and DoS. To be efficient, the Detection Engine continuously detects security attacks while processing workloads. Alerts can be stimulated during security attacks. Figure 2 describes the steps of autonomic system i.e. monitoring, analyzing and planning, and execution. During execution of workloads on scheduled resources, *Sensors* detect the value of QoS in terms of response time of task execution. Then, manager node collects information from Sensors and forwards the updated information towards analysis module.

PSEUDOCODE: SELF-PROTECTING

```
# MONITORING
START
Capture Packets
Perform parsing on captured packets
for all Packets
do
  if Packet Payload Length != Range (MIN, MAX) then
    Store packet information into log file
  end if
end for
# ANALYZING and PLANNING
# Process logs
# check for the Security Attacks
Collect all the new alerts generated by AE [Autonomic Element]
for all alerts
  do
    Perform parsing to get URL, Port and Payload detail
    Categorize data based on URL, Port and Payload
    To find largest common substring apply LCS (Longest Common Subsequence)
    Construct new signature by using payload string identified by LCS
end for
# EXECUTION
for all Signature Analyzed [SIGN_ANA]
do
  if SIGN_ANA ⊂ Existing Data then
    Signature merged to existing
  else if SIGN_ANA = Already Existing then
    'IGNORE'
  else
    Add signature as new data
  end if
end for
```

Figure 2. The steps of autonomic system, i.e. monitoring, analyzing and planning, and execution.

The *Monitoring Module* deploys security agents on different computing systems, which trace unknown attacks (using an anomaly-based detector) and known attacks (using a signature-based detector). It captures new anomalies based on existing data stored in the central database (SNORT DB). SECURE captures and detects anomalies using the Intrusion Detection System and labels it as anomalous or normal traffic data by comparing its signatures with the signatures of known attacks.

An SVM-based security agent detects the new anomalies and stores the information into the database to maintain a log about attacks. SECURE protects from security attacks: DDoS (HTTP Flood and Zero-Day Attack), Probing (NMAP and Ports sweep), U2R (Buffer Overflow and Rootkits), R2L (IMAP, Guess password and SPY) and DoS (Teardrop, SYN Flood, LAND and Smurf) as discussed in Table 1. In order to detect a security attack, only those packets will be logged which fits in the range as specified. Detection engine detects the pattern of every packet transferring through the network and compares with the pattern of packets existing in database to find the packet payload length value (range of packet). Alert will be generated if current payload length is out of range [Range (Min, Max)] and attack is detected.

The *Analyzing and Planning Module* analyzes detected attacks and generates a signature for future detection of attacks. Figure 3 shows the functions performed to generate signature.

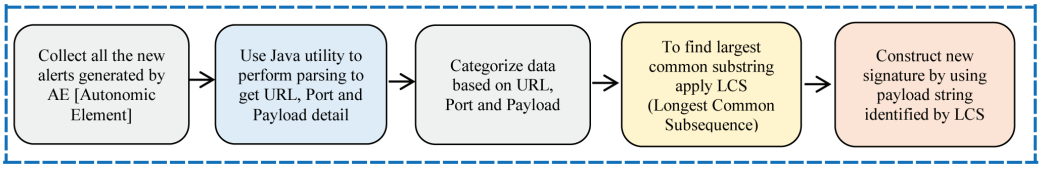


Figure 3. Process of signature generation.

For the *Execution Module*, SNORT IDS refines signatures received from previous modules and compare newly-generated signatures with existing signatures stored in SNORT database. New signatures are added to the SNORT database and new information is merged with existing signatures. Updated information among autonomic elements is exchanged by *Effector,* which communicates updated information about new alerts, rules, and policies.

PERFORMANCE EVALUATION

Figure 4 shows the experimental setup, which is used to evaluate the performance of SECURE. SNORT is a signature-based detector and works on Internet Protocol Networks to examine the real-time network for identification of malicious activity. It generates "analysis signatures" by comparing already stored signatures in the SNORT database and refines, finalizes, and stores new signatures in the SNORT database. SVM is used to detect abnormal behavior (unknown attacks). Different tools (NMAP for probing, DAVOSET for DDoS, NetCat for L2R, Hydra for R2L and metasploit for DoS) are used in this research work to launch different attacks.[1,2,3]

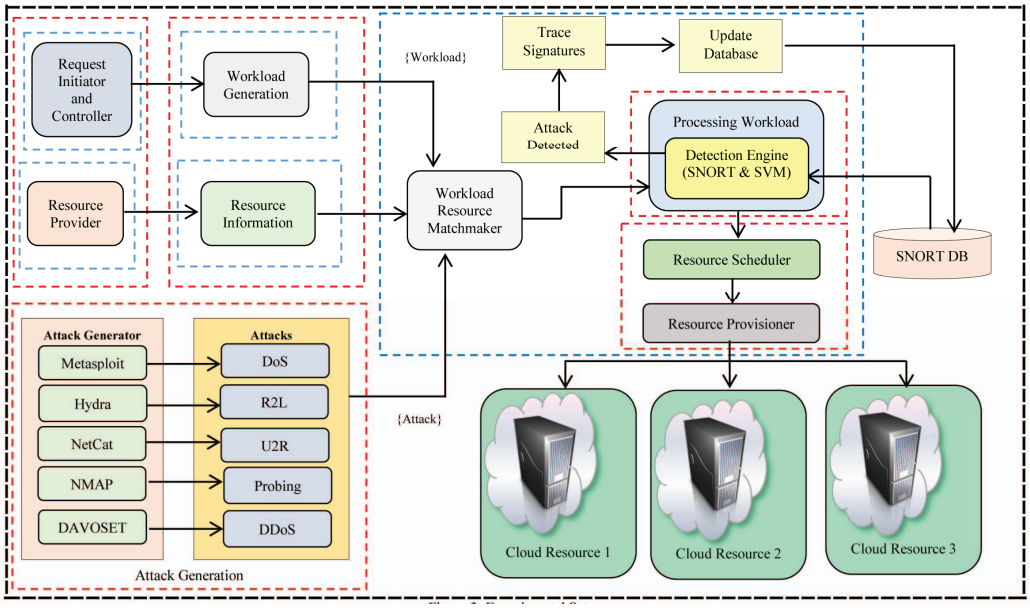


Figure 4. Experimental setup.

Experiments have been performed for five different types of attacks (DDoS, Probing, U2R, R2L and DoS) by comparing SECURE with existing security-aware resource management techniques, i.e. Self-Protection Data Scheme (SPDS).[11] Workload is conversion of larger image file of size 713 MB from JPEG format to PNG format.

Equation 1 describes the **False Positive Rate (FPR)** is described, which is the ratio of *False Positives* to summation of *False Positive and True Negatives.*

$$FPR = \frac{FalsePositives}{FalsePositives + TrueNegatives} \quad (1)$$

FPR decreases in SECURE with time and it is minimum at 50 hours as shown in Figure 5. We have considered five types of attacks (DDoS, Probing, U2R, R2L and DoS) and measured the value of FPR for each attack. The value of FPR is higher for R2L as compared to DDoS, Probing, U2R and DoS attacks.

Equation 1 describes the **Intrusion Detection Rate (IDR)**, which is the "ratio of total number of true positives to the total number of intrusions."

$$IntrusionDetectionRate = \frac{TotalNumberofTruePositives}{TotalNumberofIntrusions} \quad (2)$$

IDR considers the number of detected and blocked attacks and its value increases with respect to time. To avoid the same attack, new signatures are stored into the database continuously. For known attacks, this experiment has been conducted. Figure 6 clearly shows that SECURE gives better results as compared to SPDS in terms of IDR. Further, for more verification of SECURE, the signatures of some known attacks have been removed from the SNORT database.

Figure 7 shows that IDR is increasing with respect to time. An experiment for 144 hours has been conducted for verification of SECURE and the results show that SECURE performs better than SPDS in terms of IDR and SECURE performance is outstanding after 120 hours. Figure 8 shows the variation of IDR with respect to different numbers of workloads and different types of security attacks. With the variation of the number of workloads, the value of IDR is also rising. Figure 8 shows that SECURE performs better in probing.

Signature Generation Rate (SGR) is defined as the percentage of signatures generated over time. Figure 9 shows how SGR is calculated for both SECURE and SPDS and it shows that SECURE has higher signature generation rate than SPDS.

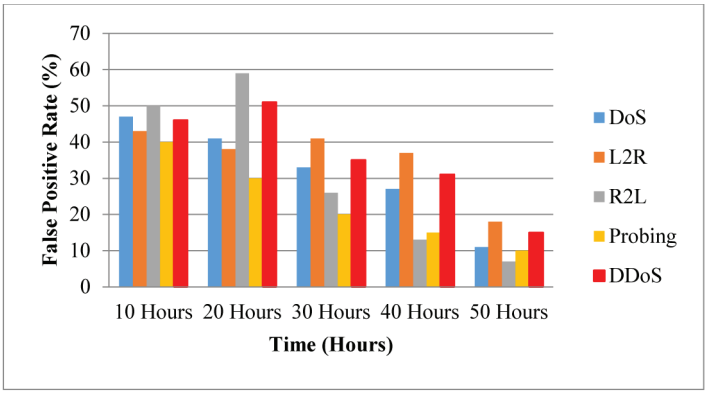


Figure 5. False Positive Rate (FPR) vs. time.

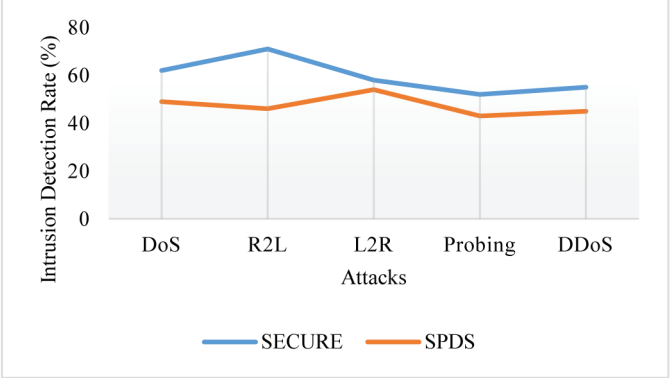


Figure 6. Intrusion Detection Rate (IDR) vs. attacks

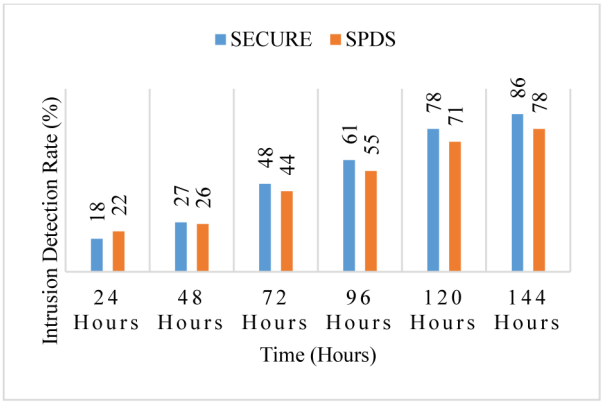


Figure 7. Intrusion Detection Rate (IDR) vs. time

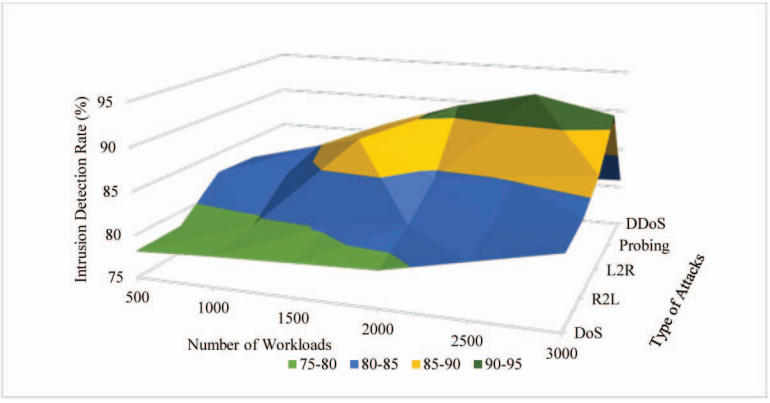


Figure 8. Intrusion Detection Rate (IDR) vs. attacks

Zero Day Attack: An Analysis

This attack denotes a security loophole in a cloud-based system that is unknown to the developer or vendor, known as zero-day attack. To perform this attack, hackers release malware before creating a patch to fix this vulnerability. To handle cloud specific zero-day attacks, a layered architecture is used to implement, which contains three different layers: i) detection layer, ii) analysis layer and iii) configuration layer as shown in Figure 10.

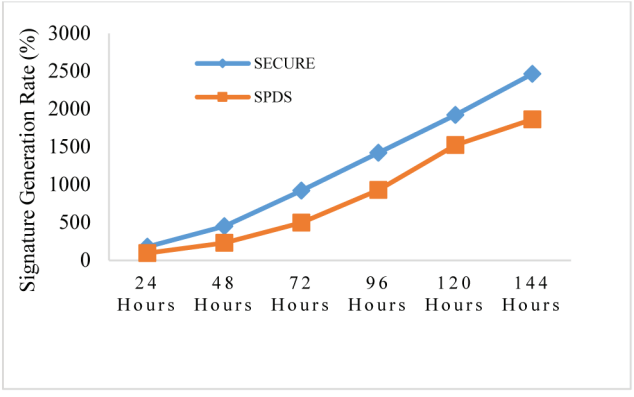


Figure 9. Signature Generation Rate (SGR) vs. time.

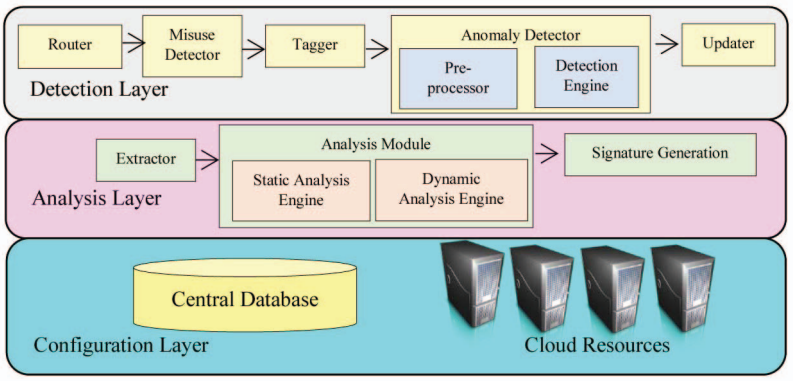


Figure 10. Layered architecture for detection of a zero day attack.

In *Detection Layer, Router* provides the input traffic data to detect unknown attacks. *Misuse Detector* models abnormal behavior because it contains the set of rules for different malicious behavior. SNORT[13] filters all the known attacks through signature matching, throws away all the known attacks, and passes the filtered network traffic. *Tagger* performs traffic tainting, in which it monitors the traffic and tags it using format *{arrival_time, source_IP, destination_IP, destination_port, Protocol}* and transfers it to *Anomaly Detector* for further processing. *Anomaly Detector* detects the unknown attacks, which are not detected by *Misuse Detector* and the pre-processor extracts features (destination bytes, source bytes, count, flag, source error rate, protocol type, and destination error rate) from tagged traffic and stores it into the log file. *Detection Engine* receives parsed traffic from the pre-processor and 1-class SVM compares this traffic with existing good traffic profile (from trusted subnet) to find zero-day attacks. *Updater* receives output from *Anomaly Detector* and forwards it to *Central Database* for future attack detection.

Analysis Layer analyzes the behavior of detected zero-day attacks. *Extractor* receives doubtful packets and parses them to detect the location using the current header length and the type of next header. The extracted data is stored as a binary file and forwarded to *Analysis Module*. Analysis Module consists of Static Analysis Engine and Dynamic Analysis Engine to merge dynamic and static functionalities to detect malware behavior. *Static Analysis Engine* analyzes the binary file to describe static features such as anti-virus scanning (to detect malicious attacks) and obfuscation (to evade detection systems). In our work, packing is used as an obfuscation technique to detect packer signatures. *Dynamic Analysis Engine* analyzes the behavior of binary files while executing and records network statistics and forwards it to next module. *Signature Generation* uses ClamAV format (*Signature = <HashString: FileSize:MalwareType>*) to generate new signatures from binary files and store them in the central database for future attack detection.

Configuration Layer contains information about cloud resources (for data processing) and the central database (to store the information about known and detected zero-day attacks). Further, Netbeans 7.0, MySql Database and Oracle Java 7 are used to evaluate the performance. The implementation system consists of different components: intranet machines, ethernet switch, IDS/IPS sensor and router. In total, 490 malware samples are used, both non-obfuscated and obfuscated, to measure the data accuracy in terms of False Positive Rate Figure 11 shows. Tools such as Metasploit, Hydra, Netcat, DAVOSET and NMAP are used to generate malware samples.[1,2,3]

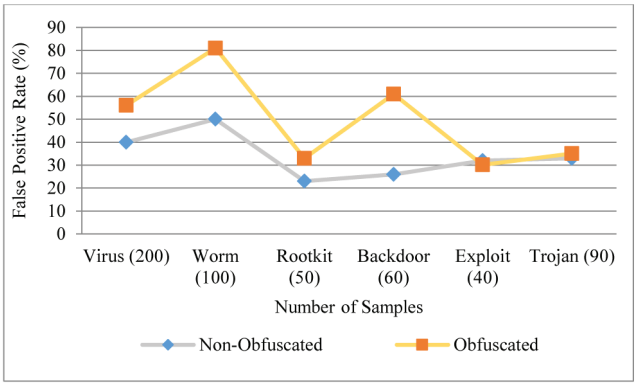


Figure 11. False Positive Rate vs. the rate of accuracy for a zero-day attack.

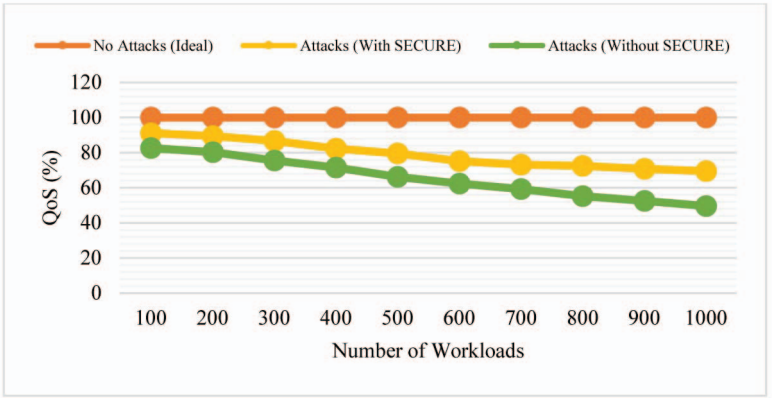


Figure 12. Impact of security on QoS (response time).

Security and QoS: Intertwined in Self-Protection of Cloud

This section analyzes the impact of Security on QoS. The *response time* is considered as a QoS parameter, which is the amount of time required to execute the workload completely and produce the required output. We have defined three different types of scenarios to measure the value of response time that Figure 12 shows. Three different scenarios follow:

- No Attack: In this scenario, there is no attack and the system executes the workloads successfully within the required response time.
- Attack Without SECURE: In this scenario, attacks occur, affecting the response time of different workloads and reducing the QoS value.
- Attack With SECURE: In this scenario, attacks occur, affecting the response time of different workloads, but SECURE improves the QoS value by detecting and neutralizing the attacks.

CONCLUSION AND FUTURE DIRECTIONS

We propose the SECURE approach for self-protection against security attacks. SECURE protects the system execution from five different types of security attacks including DDoS, Probing, U2R, R2L, and DoS, and analyzes the impact of security on QoS during the processing of user requests. Further, the performance of SECURE is tested in terms of intrusion detection rate, response time and false positive rate. Experimental results show that the performance of SECURE is better than existing techniques, which provide secure cloud based services by protecting from security attacks.

To further advance the work, we propose the following future directions:

- A practical realization of SECURE on a real cloud environment.
- SECURE can be extended to work with some other attacks also ransomware, NTP Amplification, slowloris, UDP Flood etc.
- The detection of zero-day attack can be improved by locating the source of the attack by analyzing the anomalous behavior patterns.
- Detection and analysis of multiple zero-day binaries simultaneously can improve throughput.
- Multiple execution path can be explored for effective malware analysis for detection of zero-day attack.
- Signatures for zero-day binaries can be generated in a more detailed manner using SNORT format.

ACKNOWLEDGEMENTS

We thank Arash Shaghaghi (The University of New South Wales, Australia) for comments on improving the paper.

REFERENCES

1. M.B. Mollah, M.A. Azad, and A. Vasilakos, "Security and privacy challenges in mobile cloud computing: Survey and way ahead," *Journal of Network and Computer Applications*, vol. 84, 2017, pp. 38–54.
2. S.S. Gill et al., "CHOPPER: an intelligent QoS-aware autonomic resource management approach for cloud computing," *Cluster Computing*, 2017, pp. 1–39.
3. S. Singh and I. Chana, "Q-aware: Quality of service based cloud resource provisioning," *Computers & Electrical Engineering*, vol. 47, no. C, 2015, pp. 138–160.
4. G. Qu, O.A. Rawashdeh, and D. Che, "Self-Protection against Attacks in an Autonomic Computing Environment," *International Journal of Computer Applications* (CAINE), vol. 17, no. 4, 2010, pp. 250–256.
5. A. Carpen-Amarie, "Towards a self-adaptive data management system for cloud environments," *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum* (IPDPSW 11), 2011, pp. 2077–2080.
6. A. Wailly, M. Lacoste, and H. Debar, "Vespa: Multi-layered self-protection for cloud resources," *Proceedings of the 9th International Conference on Autonomic Computing* (ICAC 12), 2012, pp. 155–160.
7. A. Sulistio and C. Reich, "Towards a Self-Protecting Cloud," *On the Move to Meaningful Internet Systems: OTM 2013 Conferences.*, Springer, 2013, pp. 395–402.
8. E. Benkhelifa and T. Welsh, "Towards Malware Inspired Cloud Self-Protection," *Proceedings of the 2014 International Conference on Cloud and Autonomic Computing* (ICCAC 14), 2014, pp. 1–2.
9. P. Manuel, "A trust model of cloud computing based on Quality of Service," *Annals of Operations Research*, vol. 233, no. 1, 2015, pp. 281–292.
10. R.D. Di Pietro, F. Lombardi, and M. Signorini, "Secure Management of Virtualized Resources," *Security in the Private Cloud*, CRC Press, 2016; doi.org/10.1201/9781315372211-14.
11. A.Y. Sarhan and S. Carr, "A Highly-Secure Self-Protection Data Scheme in Clouds Using Active Data Bundles and Agent-Based Secure Multi-party Computation," *Proceedings of the IEEE 4th International Conference on Cyber Security and Cloud Computing* (CSCloud 17), 2017, pp. 228–236.
12. S. Singh and I. Chana, "QRSF: QoS-aware resource scheduling framework in cloud computing," *The Journal of Supercomputing*, vol. 71, no. 1, 2015, pp. 241–292.
13. B. Caswell and J. Beale, *Snort 2.1 Intrusion Detection*, Syngress, 2004.

ABOUT THE AUTHORS

Sukhpal Singh Gill is a Postdoctoral Research Fellow with the University of Melbourne's Cloud Computing and Distributed Systems (CLOUDS) Laboratory. Contact him at Sukhpal.gill@unimelb.edu.au. For further information, please visit www.ssgill.in.

Rajkumar Buyya is a Redmond Barry Distinguished Professor and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is one of the most highly cited authors in computer science and software engineering worldwide (h-index=115 and 70,000-plus citations). He was recognized as a "Web of Science Highly Cited Researcher" in 2016 and 2017 by Thomson Reuters. Buyya is a Fellow of IEEE, and Scopus Researcher of the Year 2017 with an Excellence in Innovative Research Award by Elsevier for his outstanding contributions to Cloud computing. Contact him at Rbuyya@unimelb.edu.au. For further information, please visit his www.buyya.com.

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

Dealing with Cloud-Driven Enterprise Complexity

David S. Linthicum
Deloitte Consulting

According to an International Data Corporation (IDC) report, the total worldwide revenue for all cloud computing by the year 2021 should more than double the numbers seen in 2016. This gives a clear picture of the robust success of the cloud computing industry at large. More specifically, 2021 revenues are predicted to reach a startling $554 billion (https://cloudtweaks.com/2017/12/cloud-computing-predicted-drive-554-billion-2021).

The patterns of adoption within enterprises are all over the place. While private clouds were all that back in 2013, we then focused more on public clouds, and now multicloud. These changing trends have not only provided much confusion within enterprise IT as to what is the ultimate goal around the use of cloud, but they have driven much complexity into the enterprises as well.

The complexity thus far has been manageable. However, if the IDC report is correct and the market will be $554 billion by 2021, then enterprise IT complexity is likely to reach a level never before seen, with IT professionals ill prepared to deal with it. This could be a new crisis that enterprises do not see coming. Unless new approaches and new technologies are leveraged, unmanageable complexity could hinder any value coming from the use of public cloud computing.

The never-ending stream of cloud computing market reports are each more aggressive than the last, but there are a few key issues that I see in the market today:

1. **Most enterprise adoption takes place around an existing business problem, with a clear business case.** Enterprises do not adopt cloud computing because it's trendy; they need to solve real problems right out of the gate. Today, these are largely tactical, but will be strategic as cloud technology becomes more pervasive.
2. **Most enterprise adoption involves existing application migration rather than new application development.** Most enterprises that migrate applications try to do so as quickly as possible, and do not focus on "refactoring" applications to make more efficient use of native cloud services. Typically, little thought is given to what platforms are right to leverage on public clouds, they only seek platform analogs.
3. **Most enterprise adoption occurs around existing technology partners.** This includes colocation providers and managed service providers, as well as brand name enterprise technologies from companies such as IBM, Oracle, and VMware. New public cloud brands, such as Microsoft, Google, and AWS, are in there as well, of course. In-

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

deed, when enterprises talk to me about their cloud needs, they typically have a laundry list of companies they have already selected as part of their path to the cloud. It's very difficult to get them to think outside those boxes.

4. **Security and governance continue to be afterthoughts.** This, despite the fact that security is consistently listed as the number one priority for enterprises as they move to the cloud. Most rely on technology instead of planning to meet their security needs. However, most security solutions are ineffective unless implemented with a great deal of planning.
5. **Cost is not as much of a concern as we originally thought.** While most enterprises will claim to move to cloud computing for cost efficiency reasons, the reality is that most are moving due to shifts in budget. Leadership is getting tired of paying for data center upgrades and expansions each year and have set deadlines for IT to find other locations to support core IT systems. Thus, enterprise IT turned to colocation providers, managed services providers, and public clouds as the path out of owning their own data centers. Costs are rarely considered; more consideration is usually given to speed and not failing.

The resulting solutions are clouds mixed with traditional systems, mixed with other outsourcing options (e.g., colocation providers and managed services providers). This means enterprise IT must also deal with the resulting rise in complexity that will increase by 25 percent for the next five years.

Complexity is already an issue within respective businesses (50% of them, from my experience), and the additional use of public cloud-based services only makes things more complex. However, most in enterprise IT feel compelled to give cloud computing a try, with the full understanding that their IT environments are going to be more complex to build and manage. Some, however, balk at cloud adoption due to the complexity that it will bring, or has initially brought.

The enterprise architect in me would suggest that the best solution for enterprises that are already hindered by architectural complexity without the presence of cloud computing is to get their respective "acts together" before they adopt cloud computing. However, the world does not work that way. In reality, most enterprises would have to do a ton of work over many years to be perfectly ready to move to cloud-based platforms.

The root issue is the ability to manage complexity, including the addition of applications (new and old) that will run on public cloud platforms. The trick is to think in terms of replacement, not additions. Applications that exist on traditional platforms (such as LAMP in the enterprise data center) should be moved in bulk to surrogates in the cloud. Then, after some acceptance testing, those platforms should be decommissioned with extreme prejudice.

At issue is the fact that these platforms can't be shut down without all workloads being migrated off those platforms, which is almost never the case. Thus, legacy systems continue on within colocation providers or enterprises data centers until all workloads have been re-hosted. Of course, you would say that you're still dealing with fewer physical servers if you have fewer workloads. While that's correct, you still have to maintain skills to keep those platforms, as well as deal with the continued complexity. This will be the case no matter if you have one or one hundred platforms that remain.

Mistakes that many enterprises make involve moving a few applications to the cloud, which creates the need to maintain applications that run on AWS or Google, while still staring at the same hardware in the data center. Nothing changes, other than that things get more complex. At issue is always those one or two applications that are not migrated. Any cost savings made through the use of public cloud-based resources gets gobbled up by the cost and complexity of maintaining a new platform. However, this new platform happens to be within a public cloud service.

Here are a few do's and don'ts for enterprises that balk at the use of cloud computing due to the complexity it may/will bring:

- **Do enterprise architecture and overall IT planning.** Yes, that means some advanced planning, in terms of what applications and data will run where, and why. Cloud-based platforms, at the end of the day, are nothing more than additional systems you must

manage. The fewer you need to manage, the simpler the architecture, and the more likely you are to succeed longer term. These approaches and disciplines are already well defined and well known.

- **Don't push things to cloud just for the sake of pushing them to cloud.** Applications should have clearly defined benefits when they run on cloud-based platforms, with the objective of migration to decommission existing platforms so architectural complexity is actually reduced, or, at least stays about the same.
- **Do consider automation.** While cloud management platforms and service governance tools clearly add value, most enterprises don't consider them early enough in the process of moving to cloud computing. They should be systemic to the architecture and actually reduce complexity by abstracting those managed clouds away from the complexity using the "single pane of glass" approach to architecture.
- **Don't stop measuring.** Keep metrics in mind as you move to the use of cloud computing, including the ability to determine and measure cost efficiencies and overall IT efficiencies. Be prepared to get some disappointing numbers at first, and adjust the process, technology, and the architecture as needed.

The existing complexity within enterprises clearly hinders movement to cloud-based platforms. However, this movement also presents an opportunity to get this complexity under control, along with leveraging cloud computing. While many enterprises don't find this an easy path to follow, it is an opportunity to combine both the value of cloud and the value of creating and deploying more effective IT environments. But, it's going to take time, money, and a bunch of courage.

The likely path is that enterprises will understand that complexity is a problem that needs to be addressed and begin gravitating to approaches, methodologies, and technology to make this issue easier to deal with. At its heart, the problem is a lack of thinking that goes into the use of new technology, cloud-based and otherwise. If there is not forethought as to the role and configuration of this technology, complexity is the likely end-state.

That said, perhaps we're more prepared to deal with the issues of complexity now than at any time in the last 30 years of IT growth. Today's cloud management platforms and cloud service brokers are examples of tactical tools that can be employed, as well as better governance and security technology.

While this technology won't remove all of the complexity, the use of automation is a key to solving the problem. However, it's just a tactical solution as of now. If enterprises don't understand the right use cases, they could actually make things even more complex.

So, what should enterprises be doing right now? It's important to understand that this wave is coming to some degree within your enterprise. If you're prepared for it, complexity will still be a challenge but you'll likely overcome the issues. However, I feel that most enterprises won't be prepared. Those businesses could be crushed under the weight of their own complexity.

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.

IEEE computer society

SkillChoice™ Complete

Now with expanded libraries and an upgraded platform!

Valued at **$3,300!**

3,000+ online courses

MENTORSHIP

Practice Exams

28,000+ BOOKS

6,000+ VIDEOS

15,000+ *Books24x7 titles*

OVER 20x as many resources as before

One membership.
Unlimited knowledge.

Did you know IEEE Computer Society membership comes with access to a high-quality, interactive suite of professional development resources, available 24/7?

Powered by Skillsoft, the SkillChoice™ Complete library contains more than $3,000 worth of industry-leading online courses, books, videos, mentoring tools and exam prep. Best of all, you get it for the one low price of your Preferred Plus, Training & Development, or Student membership package. There's something for everyone, from beginners to advanced IT professionals to business leaders and managers.

The IT industry is constantly evolving. Don't be left behind. Join the IEEE Computer Society today, and gain access to the tools you need to stay on top of the latest trends and standards.

Learn more at **www.computer.org/join**.

ACCESS TO SKILLSOFT IS AVAILABLE WITH

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