Emergent Failures: Rethinking Cloud Reliability at Scale

Peter Garraghan
Lancaster University

Renyu Yang
University of Leeds

Zhenyu Wen
Newcastle University

Alexander Romanovsky
Newcastle University

Jie Xu
University of Leeds

Rajkumar Buyya
University of Melbourne

Rajiv Ranjan
Newcastle University

Editor:
Rajiv Ranjan
raj.ranjan@ncl.ac.uk

Since the conception of cloud computing, ensuring its ability to provide highly reliable service has been of the upmost importance and criticality to the business objectives of providers and their customers. This has held true for every facet of the system, encompassing applications, resource management, the underlying computing infrastructure, and environmental cooling. Thus, the cloud-computing and dependability research communities have exerted considerable effort toward enhancing the reliability of system components against various software and hardware failures. However, as these systems have continued to grow in scale, with heterogeneity and complexity resulting in the manifestation of emergent behavior, so too have their respective failures. Recent studies of production cloud datacenters indicate the existence of complex failure manifestations that existing fault tolerance and recovery strategies are ill-equipped to effectively handle. These strategies can even be responsible for such failures. These emergent failures—frequently transient and identifiable only at runtime—represent a significant threat to designing reliable cloud systems. This article identifies the challenges of emergent failures in cloud datacenters at scale and their impact on system resource management, and discusses potential directions of further study for Internet of Things integration and holistic fault tolerance.

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

By 2020, the first centralized exascale system will be created, comprising hundreds of thousands of nodes that provide enormous quantities of computational and storage capability. Modern cloud datacenter operation is characterized by growing system scale and diversity in workloads and their usage patterns, resource utilization, and application types with varied usage patterns. Such behavior subsequently results in diverse faults, producing failures strongly influenced by user and task behavior, resource type, workload intensity,[1] and environmental factors (such as temperature, humidity, and power) associated with cloud datacenters.

As modern cloud datacenters have continued to grow in scale and complexity, failures have become the norm, not the exception. Studies of very large-scale computing systems spanning cloud datacenters, supercomputers, high-performance computing, and clusters have demonstrated that 4% to 11% of all tasks fail,[1–3] stemming from diverse sources of software and hardware faults. This has resulted in the creation of a myriad of fault tolerance and recovery strategies focused on enhancing the availability and reliability of datacenter components, including jobs and tasks, the resource manager, physical nodes, storage, networking, and facility cooling.

Moreover, this has resulted in cloud datacenter operation manifesting emergent behavior—resultant system behavior and operation unforeseen at design time. Empirical studies of large-scale computing systems have indicated that such emergent behavior has also resulted in failure manifestation that is increasingly complex and potentially transient, stemming from correlated fault activation types.[1–4] Such failures—which we call *emergent failures*—are difficult to address because they represent "known unknown" and "unknown unknown" phenomena identified at system runtime and are often difficult to reproduce.

This is a key challenge because assumptions that underpin the design of reliable systems are defined at design time and are unable to adequately handle constantly changing error confinement boundaries and failure scenarios driven by the evolution and dynamicity of cloud datacenter operation. These failures impact all aspects of system operation from scheduling and instrumentation to workload execution, and even the fundamental assumptions that define failure propagation boundaries of components.

In this article, we discuss the nature of these emergent failures in cloud datacenters and their impact on resource management. We also outline potential areas that need to be addressed and future directions for cloud reliability research to address emergent failures.

EMERGENT FAILURE FUNDAMENTALS

The Evolution of Cloud Failures

For many decades, the creation of versatile and reliable computing systems has been achieved by defining its function and behavior (i.e., architecture, component interaction, and operational assumptions) at design time, known as the development phase in the dependability community.[5] Such an approach is wholly intuitive. To create a desired system, it is necessary to first explicitly define its respective behavior to implement appropriate mechanisms ensuring its dependability.

In the context of reliability, systems are defined via expert analysis and the specification of assumptions pertaining to fault and failure types, error propagation across components and system boundaries, the necessary fault tolerance and recovery strategies, and the respective coverage required to effectively address selected failures.

Because of the potential impact on system performance and cost, it is often considered viable to consider only a limited scope of fault types and failure coverage owing to diminishing returns in fault prevention. For example, a system designer can decide not to commit considerable engineering effort to tolerate incredibly rare yet minor failures. Such an approach is driven by the need to reduce the complexity of system design and to localize error recovery.

When failures do manifest outside the confines of a set of defined assumptions, maintenance is required to conduct system repair and modification to address the fault's root cause. In cloud datacenters, as in any other complex system, it is inevitable that it is impossible to cover all types of faults and failures that could occur. However, cloud datacenters are and will continue to be frequently exposed to conditions and scenarios that result in a large variety of faults and failure scenarios that were not envisioned at design time.

Dynamicity and Heterogeneity

A positive correlation exists between the resource type, workload intensity, and failure rate.[1] As workload dynamicity is an intrinsic property of cloud computing, it is difficult to forecast the precise conditions that precipitate failure. Such dynamicity is not solely limited to the workload; it also encompasses server power consumption, network traffic, and environmental conditions (e.g., temperature hotspots).

This problem becomes compounded when these factors are combined. A workload can execute on a diverse range of system architectures (refreshed by a datacenter approximately every nine months), microprocessor types (CPU, GPU, NPU [neural processing unit], etc.), network configurations, and cooling technologies (air or liquid). Such heterogeneity allows cloud datacenters to offer a variety of services while minimizing the likelihood of common-mode failure. However, it does so at the expense of increasing the system's exposure to different fault types and component interactions for which the system was not originally designed.

Scale and Complexity

Cloud datacenters operating on a massive scale are exposed to more frequent and complex failure scenarios. Owing to an increase in potential system states and in the complexity of component interactions, it can be difficult to ascertain the precise root cause of failure manifestation and its dependencies on components across the system. Datacenter operators frequently encounter scenarios in which hundreds of failure event notifications from different components are eventually traced to a root cause in a seemingly unrelated component event. Moreover, a system with more components intuitively experiences higher failure frequency. Assuming identical mean time between failure (MTBF) of components, a 10,000-node datacenter will encounter more frequent component failures compared to a 1,000-node datacenter.

That is not to say that these conditions alone have resulted in highly unreliable systems. If that were the case, existing cloud datacenters would not operate. However, it is an indication of two growing trends in large-scale systems that directly threaten their reliability. First, as cloud datacenters continue to evolve in terms of their scale, dynamicity, heterogeneity, and complexity, the manifestation of emergent failures is also increasing. Second, it is increasingly challenging to ensure system reliability when human-defined design assumptions for fault types, propagation, and fault tolerance and recovery strategies might not be appropriate for the current operational conditions of cloud datacenters.

Potential Causes of Emergent Failures

Emergent failures are types of failure that are manifested within constantly changing error propagation boundaries intersecting hardware and software components, have the potential to be transient, and are identifiable only at system runtime. There exist various examples of emergent failure phenomena in large-scale cloud datacenters, with their effects ranging from minor system degradation to catastrophic facility outage.

Performance Interference

Virtualization encapsulates functionality to construct well-defined fault assumptions for virtual machines (VMs). However, VMs in multitenant servers transparently share the same underlying resources. This results in performance interference between VMs and daemon processes within the server, increasing late-timing failure likelihood for interactive tasks. The challenge is that such phenomena vary considerably based on workload and hardware heterogeneity, and that VMs are not designed to mitigate effects outside of their operational boundaries.

Stragglers

Stragglers are also known as *tailing behavior*, whereby a subset of a job executes abnormally slower compared to typical tasks,[4] resulting in late-timing failures for any jobs that enforce time-related service-level agreements (SLAs). It has been demonstrated that 5% of task stragglers impact more than half of the jobs in a datacenter.[3]

Understanding and mitigating stragglers is an open challenge in the distributed-systems community. This challenge pertains to detection and forecasting because of stragglers' transient nature and manifestation. This problem potentially stems from a variety of sources, including daemon processes, data skew, resource contention, component failures, server hotspots, energy management, or a combination of any of these.

"Competing" Fault Tolerance

Fault tolerance is designed assuming defined layers of abstraction between components. For example, a subsystem comprising multiple components (such as a VM containing an OS) can activate a particular fault-tolerance strategy to ensure that a service adheres to specified availability and reliability requirements. However, because such components are created independently from other system components, the fault-tolerance strategy for one subsystem can unknowingly impact the service of components outside its operational boundary. Creating a VM replica can result in increased performance interference and stragglers in other VMs, or increase server temperature, resulting in a hotspot requiring task eviction, and so on.

Cascading Recovery

Ironically, recovery strategies in cloud datacenters can also result in emergent failure manifestation. A well-documented case study of such failures is the 2017 Amazon outage. This outage resulted from Amazon S3's substantial growth over the previous few years, such that the process of restarting S3 services and running safety checks to validate metadata integrity took longer than expected. These delays resulted in an unintended failure cascade between recovery strategies as other AWS (Amazon Web Services) services impacted by this event also began recovering. These services accumulated a backlog of work during S3 disruption and themselves required additional time to recover. The scale of this problem was identified by the Argonne National Laboratory, which stated that such an outage demonstrated that interdependencies between datacenters and network providers are not well understood, which further compounds the challenge of creating resilient infrastructure.[6]

Emergent failures can also have hardware and software causes, including, but not limited to, channel overloading, power shortages, incorrect kernel caching, unpredictably invalid memory access due to wild or dangling pointers, unexpected race conditions in concurrent threads, kernel or human-made bugs, and incorrect configurations. The key idea underpinning these failures is that they are a by-product of emergent operational behavior unanticipated at system design.

Existing fault tolerance and recovery mechanisms are unable to alter their operation and coverage in response to any of these causes in cloud datacenters, without manual intervention after failure occurrence. Thus, emergent failures are frequently omitted from most fault tolerance and recovery design owing to their complexity. However, these types of failures will become more prominent as cloud datacenters grow in scale and complexity and become even greater with the increased prominence of the Internet of Things (IoT) and fog computing.

EMERGENT FAILURES IN RESOURCE MANAGEMENT

Resource management is a fundamental aspect of cloud datacenter operation facilitated by deployment of a resource manager (such as Kubernetes, Fuxi, YARN, and Mesos) that orchestrates machine resources, applications, and users along with scheduling and monitoring the execution of jobs and tasks. Modern cloud datacenters attempt to ensure that all submitted jobs are successfully scheduled (in reality, 99.999%), executed, and completed without loss of correct service perceivable by the customer. The resource scheduler attempts to achieve this by monitoring machine health, finding available resources for pending tasks, deploying binaries and launching workloads, restarting failed jobs, and restoring state during failover.

Specifically, failures in resource managers are predominately the result of (i) *time-out* caused by the overall latency aggregated from different service calls for jobs (interactive jobs that experience a slowdown and have a timing SLA imposed), and (ii) *component hang or crash* due to resource exhaustion (a faulty service or component results in insufficient resources for regular request handling of other tasks).

The challenge is that these causes are increasingly the result of emergent failures. As shown in Figure 1, resource managers are required to provide resources (compute, storage and network) to increasingly various levels of abstractions (VMs, containers, batch jobs, object storage, etc.) within large-scale dynamic cloud datacenter environments, thus making it difficult to capture failures that transcend established component boundaries.

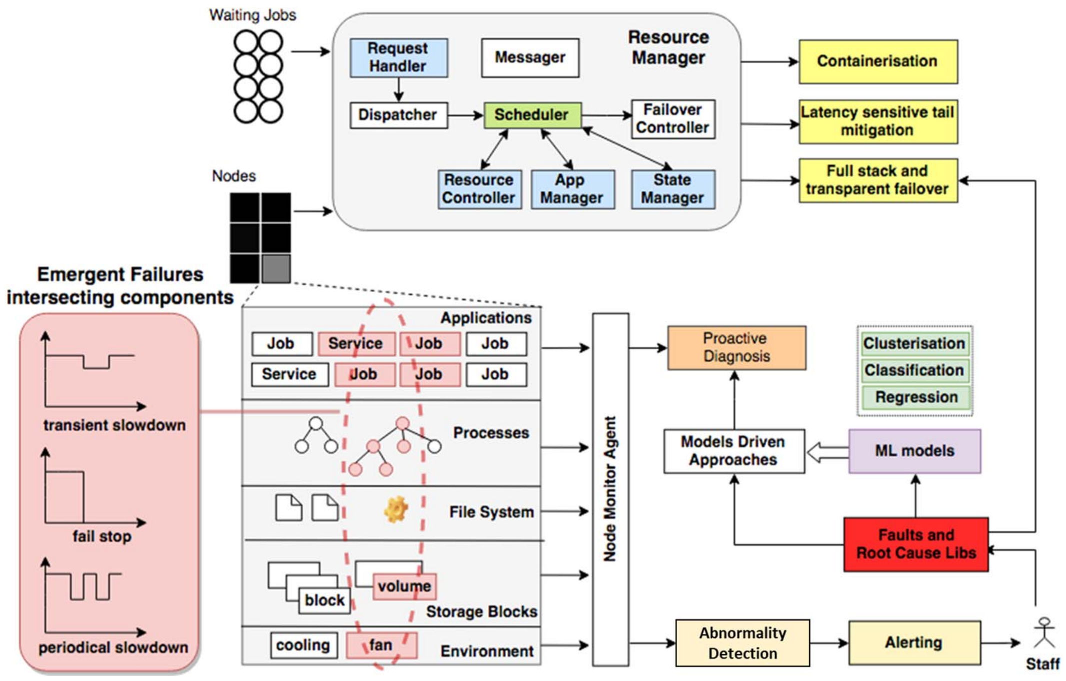

Figure 1. Emergent failure manifestation in cloud datacenter resource management.

We discuss three different perspectives as to how emergent failures affect resource management, as well as how to alleviate their effects: architectural factorization to isolate failures and reduce their propagation, runtime monitoring to detect anomaly behavior in a timely manner, and instrumentation for proactive prevention and tolerance.

Containerized Architecture Rethinking

Architectural Evolution

The centralized resource manager architecture[7–11] is a monolithic system that contains all functional components (request handler and dispatcher, communication messenger, state manager, decision maker, etc.) contained in a single process or multiple processes. Although decentralized scheduling[12,13] can dispatch such functionality to distributed components in a loosely coupled manner, they are still logically monolithic from the holistic view. There is an increasing likelihood that emergent failures will manifest from memory exhaustion (due to faulty components), resulting in an overall crash–stop failure, unresolved deadlock in the decision maker resulting in the slowdown of request handling, and late-timing state mismatch in the state manager leading to the scheduling conflicts.

As a result, there has been a need to leverage submodulization and containerization of the datacenter resource manager.[14] For example, the resource manager master scheduler should be able to function in the face of various failures. To orchestrate and run containers, other system components such as container clustering, networking, and automated deployment and monitoring are required. For instance, Kubernetes schedules any number of container replicas across a group of nodes. Increasingly, Kubernetes components or external plugins that would traditionally be deployed within the bare metal machines are instead deployed and maintained within containers themselves to increase management flexibility.

Fault Isolation and Propagation Prevention

Resource exhaustion[15] is a leading root cause of crash–stop or timing failures in system components. It can be caused by either a failure in a single component or other faulty and nonfaulty component behavior outside the defined system boundaries. For example, a service that experiences high latency (due to stragglers or crash failures in the network) can result in communicating services experiencing resource exhaustion. Performance interference between tasks in the same physical node results in performance degradation and resource exhaustion in other tasks.

System designers attempt to mitigate such propagation by leveraging container-based mechanisms and cgroup restrictions whose operation is dictated by quantitative quality-of-service modeling to define the conservatively least resource boundary of each job group. However, determining the most appropriate parameters (and, importantly, how they should evolve in response to changes in operational context) is an open research challenge.

Cloud Monitoring—Timely Detection and Alerting

Robust Monitoring and Alerting

At increased system scale, real-time health checking, load measurement throughput, and application-specific errors become increasingly important. However, an outstanding issue is how to effectively monitor system health when considering the sheer volume and variety of hundreds of millions of potential system metrics. When exposed to the manifestation of emergent failures that can be caused by monitoring itself, traditional static threshold-based monitoring and alerting are insufficient. A human-defined threshold might be useful to enact automated decision making and alerting on-call technical staff. However, it might encounter difficulties in terms of false negatives and false positives that might change in response to system usage.

Therefore, a robust anomaly detection mechanism whose sensitivity can be appropriately tuned in accordance with the current operational context of the system is required. A potential means to achieve this is by leveraging adaptive learning of monitoring and detection parameters that considers different periodicities, parameter types, and parameter values. However, how to generate and exploit streaming metrics to recognize outliers is intricately challenging due to the dilemma system monitors face—selectively using partial metrics to enact fast (yet imprecise) decisions, or exploiting a large number of metrics for more precise (yet slow) decision making.

Preventive Performance Diagnosis

In reactive solutions, a faulty running service is halted to ascertain what conditions led to emergent failure manifestation to enact necessary maintenance (which has been demonstrated to be ineffective for dealing with stragglers[4]). In contrast, a proactive diagnosis would ensure that user services are minimally affected. Monitoring as many components as possible is likely to support failure prediction. However, in practice, not all components can be monitored, owing to the sheer volume of data required to be collected, transmitted, and calculated.

Taking into account information pertaining to hardware and environmental factors such as fan speed or temperature, it is highly desirable to explore the failure root causes and investigate the interactions of system components in failures caused by multiple faults. However, it is extremely difficult to articulate the root causes at runtime, owing to uncontrollable and intrinsic system factors. Statistical correlation among metrics can facilitate rapidly finding root causes and determining the most effective handler.

Component self-diagnosis is also beneficial to the system instrumentation. For example, understanding and leveraging node performance is critical for straggler mitigation and workload placement. Performance refers to a node's ability to execute parallel applications and hold containerized services. Machine-learning techniques such as classification and regression (e.g., random forests, gradient-boosting trees) might be one means to achieve this. Through classifying nodes into different categories and predicting the corresponding performance category with high accuracy, the scheduler can rank nodes and select suitable nodes to launch latency-sensitive tasks. This process avoids assigning speculative tasks onto nodes that are likely to be in a weak performance state.

Cloud Scheduling and Instrumentation: Prevention and Tolerance

Emergent failure aware design should permeate into each step and component of the cloud scheduler. To reduce scheduling downtime, the system design should not have a single point of failure. The ultimate vision is to realize a zero-downtime scheduler system.

Latency-Oriented Tail Mitigation Based on Redundancy

Modern cluster schedulers must deal with both latency-sensitive requests and computationally intensive tasks (e.g., long-running HTTP services and periodic cron jobs). Redundancy is the fundamental technique used to enhance component reliability of hardware, software, and data storage. On the basis of multireplica component deployment, identical components can be deployed.

The replication controller is typically used to track and record the health status of replicated components. The controller should guarantee the number of provisioned replicas at any given moment. That is, the controller should launch a new replica if a component is killed or becomes inaccessible. For instance, in Kubernetes, the ReplicationController can autoscale and manage microservices on the basis of resource utilization or a fixed lower or upper limit of the expected number of replicas.

For computationally intensive tasks, the most common means to resolve stragglers is speculative execution relying on idempotency. However, a lack of coordinated fault tolerance between components leads to an emergent failure whereby such an action results in increased resource contention, leading to cascading latencies for new tasks. Stragglers arise even more frequently in learning systems and distributed optimization because performance is significantly throttled by slow communication and computation. The idempotency is invalid owing to the shared states. Machine-learning scenario-specific mitigations such as data encoding with built-in redundancy in certain linear-computation steps[16] enable the system to complete computation to tolerate the effects of stragglers.

User-Transparent Failover and Fault Conversion

The system designer attempts to design the resource scheduler so that it can perform failover and self-healing (autonomous recovery) of all components, unperceived by the customer. An important consideration for conducting failover is state recovery that prominently leverages caching or checkpointing. Intermediate states or returned results from stateless services can be cached so that the majority of services can continue operating during intermittent failures in any related components. For more critical data or state (such as runtime memory bitmap and register values), checkpointing can be leveraged to create snapshot backups of current system states.

Although this strategy is effective for recovering from incorrect state and data loss, the checkpointing itself is often considerably large. Checkpointing in a 1,000-node datacenter cluster in Alibaba over 24 hours has been reported to generate a 1.7 Gbyte checkpoint (and in high-performance computing, checkpointing can take hours to complete),[17] and as demonstrated by the 2017 Amazon outage, checkpointing can unknowingly manifest as an emergent failure itself.

Therefore, we believe that new approaches are required for checkpointing to function at scale, such as combining hard-state backup and soft-state inference.[17] However, because emergent failures cannot be anticipated, it is essential to enable the finite-state machine of system faults to be more able to adapt in accordance to detected system faults. For example, this could be conducted by automatic transformation of an emergent fault mode into that of a known fault mode classification that can then accordingly tackle faults through established approaches. Once a fault is determined, the components or devices (such as storage blocks or network interface controllers) that lead to performance degradation could be temporarily isolated or removed during system failover.

RETHINKING BEYOND CLOUDS

Holistic Fault Tolerance and Recovery

Holistic fault tolerance (HFT) has been recently introduced and could be an effective approach for handling emergent failures. HFT relies on cross-cutting components for system recovery tailored to the specific error detected and the appropriate recovery strategy for execution. The recovery region strictly involves system components that need to be involved for recovery for a given error. These components, which could be located at different layers, subsystems, packages, nodes, etc., are involved in a coordinated recovery. This approach makes it possible to reduce system complexity to address complex failure recovery scenarios.

For example, in order to address the challenges of performance interference, it could be possible to coordinate two VMs on the same physical node. When one VM fails to adhere to timing requirements, HFT could consider performing coordinate recovery by leveraging components in both VMs. This could be facilitated by the hypervisor altering its scheduling to provide more of a CPU to a particular VM, and then measuring the resultant delays in both VMs to ensure satisfactory levels of CPU share. If the two VMs are unable to do so, the hypervisor itself would then need to make this change. If this is not possible, then a wider decision to evict and reschedule the VM would be required that incorporates the resource manager.

IoT Integration

The presence of emergent failures is not solely confined to cloud datacenters; they can manifest prominently in any large-scale computing system including emerging fog- and edge-computing models supporting IoT applications. These systems are particularly susceptible to emergent failures for many of the reasons given for clouds—a dynamic and unpredictable assortment of interconnected virtual and physical devices. A key difference is that IoT, fog-computing, and edge-computing systems exhibit a high degree of join–leave behavior not found within cloud computing due to their centralized nature.

If the system boundaries of interconnected components are constantly changing owing to their usage and device composition, it is intuitive to assume that rigid fault-tolerance strategies that are designed independently from the operational context of the greater system will be increasingly infeasible. Such system environments will also likely result in “fluid” error confinement areas for a set of components (e.g., constantly changing). Hence, we believe a future research direction will be to investigate how to autonomously determine the optimal fault tolerance and recovery mechanism for a given system context.

CONCLUSION

In this article we discuss the rise of emergent failures: a growing problem toward ensuring reliability in cloud datacenters and all future computing systems at scale. A central issue to address is how to determine effective fault tolerance and recovery strategies when assumptions that define fault types and failure scenarios are constantly changing due to cloud datacenter dynamicity, complexity, and heterogeneity between interacting components. Two potential ways to address this issue are (i) rethinking the nature of system abstraction allowing for holistic fault tolerance that cross-cuts coordination of components, and (ii) exploring the concept of adaptive fault tolerance in response to current and forecasted operational scenarios. Moreover, further study is required by the research community to study the relationship between cloud datacenter operation and emergent failure manifestation beyond coarse-grained analysis and observation, and toward creating models that precisely capture system conditions that lead to failure.

ACKNOWLEDGMENTS

This work is supported by the UK Engineering and Physical Sciences Research Council (EP/P031617/1) and the National Key Research and Development Program of China (2016YFB1000103). Renyu Yang is the corresponding author for this article.

REFERENCES

1. B. Schroeder and A.G. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *Proceedings of the International Conference on Dependable Systems and Networks* (DSN 06), 2006, pp. 249–258.
2. P. Garraghan et al., "An Analysis of Failure-Related Energy Waste in a Large-Scale Cloud Environment," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, 2014, pp. 166–180.
3. P. Garraghan et al., "Straggler Root Cause and Impact Analysis for Massive scale Virtualized Cloud Datacenters," *IEEE Transactions on Services Computing*, 2016.
4. J. Dean and L.A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, 2013, pp. 74–80.
5. A. Avižienis et al., "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, 2004, pp. 11–33.
6. M. Thompson, "Amazon S3 Outage Highlights Resilience Issues with Cloud Infrastructure," Argonne National Laboratory, 2017; https://coar.risc.anl.gov/amazon-s3-outage-highlights-resilience-issues-cloud-infrastructure.
7. A. Verma et al., "Large-scale cluster management at Google with Borg," *Proceedings of the Tenth European Conference on Computer Systems* (EuroSys 15), 2015, p. 18.
8. B. Burns et al., "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, 2016, p. 10; https://queue.acm.org/detail.cfm?id=2898444.
9. B. Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (NSDI 11), 2011, pp. 295–308.
10. V.K. Vavilapalli et al., "Apache Hadoop YARN: yet another resource negotiator," *Proceedings of the 4th annual Symposium on Cloud Computing* (SOCC 13), 2013, p. 5.
11. Z. Zhang et al., "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, 2014, pp. 1393–1404.
12. E. Boutin et al., "Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing," *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* (OSDI 14), 2014, pp. 285–300.
13. X. Sun et al., "ROSE: Cluster Resource Scheduling via Speculative Over-Subscription," *IEEE 38th International Conference on Distributed Computing Systems* (ICDCS 18), 2018; doi.org/10.1109/ICDCS.2018.00096.

14. M. Sossa and R. Buyya, “Container-Based Cluster Orchestration Systems: A Taxonomy and Future Directions,” *ArXiv*, July 2018; https://arxiv.org/abs/1807.06193.
15. B. Mauer, “Fail at Scale,” *ACM Queue*, vol. 13, no. 8, 2015, p. 30.
16. C. Karakus et al., “Straggler Mitigation in Distributed Optimization Through Data Encoding,” *Advances in Neural Information Processing Systems*, 2017, pp. 5434–5442.
17. R. Yang et al., “Reliable computing service in massive-scale systems through rapid low-cost failover,” *IEEE Transactions on Services Computing*, vol. 10, no. 6, 2017, pp. 969–983.

ABOUT THE AUTHORS

Peter Garraghan is a lecturer in distributed systems at Lancaster University. His research interests encompass massive-scale distributed systems, dependability, resource management, and energy efficiency. Garraghan received a PhD in computer science from the University of Leeds. Contact him at p.garraghan@lancaster.ac.uk.

Renyu Yang is a research fellow at the University of Leeds and an R&D scientist at Edgetic. His research interests include massive-scale distributed systems, resource scheduling, and dependability. Yang received a PhD in computer science from Beihang University. He is the corresponding author. Contact him at renyu.yang@edgetic.com.

Zhenyu Wen is research fellow at Newcastle University. His research interests include the Internet of Things, distributed systems, big data analytics, and computer networks. Wen received a PhD in cloud computing from Newcastle University. Contact him at zhenyu.wen@newcastle.ac.uk.

Alexander Romanovsky is a chair professor of computing science at Newcastle University. His research interests include fault tolerance and system dependability. Romanovsky received a PhD in computer science from St. Petersburg State Technical University. Contact him at alexander.romanovsky@ncl.ac.uk.

Jie Xu is a chair professor of computing, the leader of a Research Peak of Excellence, and the head of the Distributed Systems and Services group at the University of Leeds. He’s also a cofounder of Edgetic. His research interests include large-scale distributed computing and dependability. Xu received a PhD in advanced fault-tolerant software from Newcastle University. Contact him at j.xu@leeds.ac.uk.

Rajkumar Buyya is a Redmond Barry Distinguished Professor at the University of Melbourne. His research interests include the cloud, parallel computing, resource management, and reliable and energy-efficient datacenters. Buyya received a PhD in computer science from Monash University. Contact him at rbuyya@unimelb.edu.au.

Rajiv Ranjan is a chair professor of computing science and Internet of Things at Newcastle University. His research interests include the Internet of Things and big data analytics. Ranjan received a PhD in computer science and software engineering from the University of Melbourne. Contact him at raj.ranjan@ncl.ac.uk.