Article:
Garraghan, P, Yang, R, Wen, Z et al. (4 more authors) (2018) Emergent Failures: Rethinking Cloud Reliability at Scale. IEEE Cloud Computing, 5 (5). pp. 12-21. ISSN 2325-6095

https://doi.org/10.1109/MCC.2018.053711662

Emergent Failures: Rethinking Reliable Cloud Datacenters at Scale

Peter Garraghan
Lancaster University,
United Kingdom

Renyu Yang
University of Leeds,
United Kingdom

Zhenyu Wen
Newcastle University,
United Kingdom

Alexander Romanovsky
Newcastle University,
United Kingdom

Jie Xu
University of Leeds,
United Kingdom

Rajkumar Buyya
University of Melbourne,
Australia

Rajiv Ranjan
Newcastle University,
United Kingdom

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. This has held true for every facet within the system, encompassing applications, resource management, the underlying computing infrastructure, and environmental cooling. Thus, the Cloud computing and dependability research communities have exerted considerable efforts towards enhancing the reliability of system components against various software and hardware failures. However, as these systems have continued to grow in scale, heterogeneity and complexity resulting in the manifestation of emergent behaviour, so too have their respective failures. Recent studies of production Cloud datacenter indicate the existence of complex failure manifestation which existing fault tolerance and recovery strategies are ill-equipped to effectively handle and can even be responsible for such failures themselves. These emergent failures – frequently transient and only identifiable at run-time – represent a significant threat towards designing reliable Cloud systems. This work identifies the challenges of emergent failures within Cloud datacenters at scale, their impact upon system resource management, and discusses potential directions of further study for IoT integration and holistic fault tolerance.

Introduction

By 2020, the first centralized Exascale system will be created, comprising hundreds of thousands of nodes that provisions enormous quantities of computational and storage capability. Modern Cloud datacenter operation is characterized by growing system scale and diversity within workload usage patterns, resource utilization, and application types. Such behaviour subsequently results in diverse fault activation producing failures strongly influenced by user and task behaviour, resource type, workload intensity [1], and environmental factors (temperature, humidity, power).

As modern Cloud datacentres have continued to grow in scale and complexity, failures have become the norm, and not the exception. Studies of very large-scale computing systems spanning Cloud datacenters, supercomputers, HPC, and clusters have demonstrated that 4-11% of all tasks fail [1][2][3] stemming from diverse sources of faults in software and hardware. This has resulted creating 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 Cloud in datacenter operation manifesting *emergent behaviour* - system operation unforeseen at design time. Empirical study of large-scale computing systems have indicated that such emergent behaviour has also resulted in failure manifestation that is increasingly complex and transient in nature stemming from correlated fault activation types [1]-[4]. These failures types – which we term emergent failures – are difficult to address as they represent “known unknown” and “unknown unknown” phenomena identified at system run-time and are oftentimes difficult to reproduce. This is a key challenge as assumptions that underpin designing 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, workload execution, and even the fundamental assumptions that define failure propagation boundaries of components.

In this work, we discuss the nature of these emergent failures within Cloud datacenters and their impact upon resource management. Moreover, we outline potential areas that need to be addressed and future direction for Cloud reliability research to address emergent failures.

Emergent Failure Fundamentals

The Evolution of Cloud Failures

For many decades within computer science, the creation of a computer system is achieved by defining its *function* and *behaviour* (i.e. architecture, component interaction, and operational assumptions) at design time known as the *development phase* within the dependability community [5]. Such an approach is wholly intuitive – to create a desired system, it is necessary to first explicitly define its respective behaviour to implement appropriate mechanisms ensuring its dependability. Within the context of reliability, systems are defined via expert analysis and specification of assumptions pertaining to faults and failures types, error propagation across components and system boundaries, necessary fault tolerance and recovery strategies, as well as their respective coverage required to effectively address selected failures. Due to the potential impact upon system performance and cost, it is often considered viable to only consider a limited scope of fault types and failure coverage (e.g. 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 localise 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 root-cause. Within Cloud datacenters, it is inevitable that it is not possible to cover all types of faults and failures that may potentially manifest. However, present day and future Cloud datacentres are frequently exposed to conditions and scenarios that result in a large variety of faults and failure scenarios which are not envisioned at design time:

***Dynamicity & Heterogeneity**.* There exists a positive correlation between workload 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 workload, but encompasses server power consumption, network traffic, and environmental conditions (e.g. temperature hotspots). This problem becomes compounded when these factors are combined; workload can execute within a diverse range of server architectures (refreshed by a datacenter approximately every nine months), microprocessor types (CPU, GPU, NPU, etc.), network configurations, and cooling technologies (air or liquid). While such heterogeneity is advantageous for Cloud datacenters to minimize likelihood of common-mode failure, it does so at the expense of increasing its exposure to different fault types and component interactions that the system is not originally designed for.

Scale & Complexity. Cloud datacenters operating at massive-scale are exposed to more frequent and complex failure scenarios. Due to an increase in potential system states and complexity in component interactions, it is difficult to ascertain the precise root-cause of failure manifestation and its dependencies with components across the system. Datacenter operators frequently encounter scenarios whereby hundreds of failure event notifications from different components are eventually traced to a root-cause within a seemingly non-related component event. Moreover, a system with a greater number of components intuitively experiences higher failure frequency. If assuming identical Mean Time Between Failure (MTBF) of components, a 10,000 node datacenter will encounter more frequent component failures in comparison 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 indicative of two growing trends within 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 has also increased. Second, it is increasingly challenging to ensure system reliability when human-defined design assumptions for fault types, propagation, as well as fault tolerance and recovery strategies may not be appropriate for the current operational conditions of the Cloud datacenter.

Potential Causes of Emergent Failures

Emergent failures are types of failure characterized by their transient nature, manifestation within constantly changing error propagation boundaries intersecting hardware and software components, and are only identifiable at system run-time. 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*.* Virtualisation encapsulates functionality to construct well-defined and fault assumptions for Virtual Machines (VMs). However, VMs in multi-tenant 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 phenomenon varies considerably based on workload and hardware heterogeneity, and that VMs are not designed to mitigate effects outside of its operational boundary.

Stragglers: Also known as tailing behaviour 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 SLAs. It has been demonstrated that 5% of tasks stragglers impact over half of the jobs within a datacentre [3]. Understanding and mitigating stragglers is an open challenge in the distributed systems community pertaining to detection and forecasting due to their transient nature and manifestation stemming from a wide variety of potential sources from daemon processes, data skew, resource contention, component failure, server hotspots, energy management, and a combination of any of these causes together.

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

Cascading Recovery. Ironically, recovery strategies within Cloud datacenters can also result in emergent failure manifestation. A well-documented case study of such failures is the Amazon 2017 outage. This outage was resultant of S3 experiencing substantial growth over the past few years, whereby the process of restarting S3 services and running safety checks to validate metadata integrity took longer than expected. Such delays resulted in unintended failure cascade between recovery strategies as other AWS services impacted by this event also began recovering, and these services accumulated a backlog of work during S3 disruption that themselves required additional time to recover. The scale of this problem has been identified by The Argonne National Laboratory stating that such an outage demonstrated that interdependencies between datacenter and network providers are not well understood, and further compounds the challenge of creating resilient infrastructure.

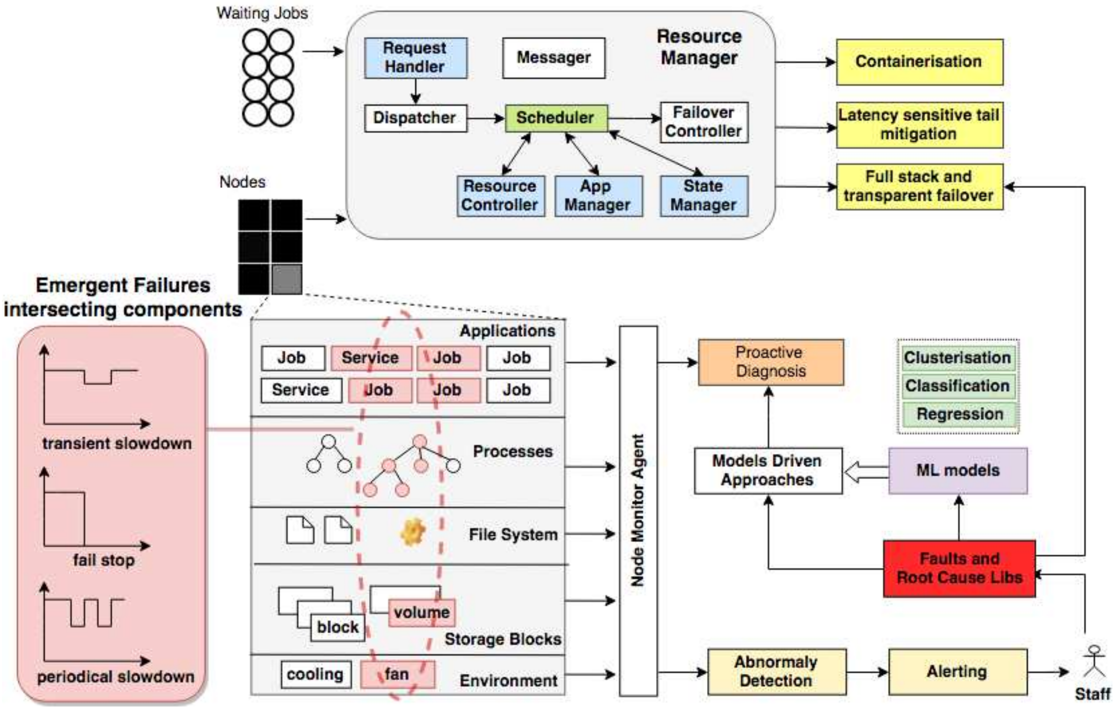

Figure 1. Emergent failure manifestation in Cloud datacenter resource management

Emergent failures can also stem from hardware and software reasons including but not limited to: channel overloading, power shortage, incorrect kernel caching, unpredictably invalid memory access due to wild/dangling pointers, unexpected race conditions within concurrent threads, kernel or human-made bugs, incorrect configurations and so forth. The key principle idea underpinning these failures is that they are a by-product of emergent operational behaviour 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 within Cloud datacenters without manual intervention after failure occurrence. Thus, emergent failures are frequently omitted from the majority of fault tolerance and fault recovery design due to their complexity. However, these types of failures will become more prominent as Cloud datacenter grow in scale and complexity and become even greater with a rise IoT and Fog computing prominence.

Emergent Failures in Resource Management

Resource management is a fundamental aspect of Cloud datacenter operation facilitated by deployment of a resource manager (Kubernetes, Borg, Fuxi, YARN, Mesos, etc.) that orchestrates machine resources and monitors 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 workload, restart failed jobs and restore state during failover.

Specifically, failures within resource managers are predominately resultant of (i) time-out caused by the overall latency aggregated from different service calls for jobs (i.e. interactive jobs that experience slowdown and have a timing SLA imposed), and (ii) component hanging or crash due to resource exhaustion (i.e. faulty service or component results in insufficient resources for regular request handling of other tasks). The challenge is that these causes are increasingly resultant of emergent failures. As shown within Fig. 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 capturing failures which transcend established component boundaries.

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

Containerized Architecture Rethinking

Architectural evolution - The centralized resource manager architecture [6-10] is a monolithic system which contains all functional components (request handler and dispatcher, communication messenger, state manager, and decision maker, etc.) contained within single or multiple processes. Although decentralized scheduling [11][14] can dispatch such functionality to distributed components in a loose-coupled manner, they are still logically monolithic from the holistic view. There is an increasing likelihood that emergent failures manifest from memory exhaustion (due to faulty components) resulting in the overall crash and stop, unsolved 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 sub-modulization and containerization of the datacenter resource manager. 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. To benefit from containerization and increase the management flexibility, increasing Kubernetes components or external plugins are deployed and maintained within containers.

Fault isolation and propagation prevention - Resource exhaustion [12] is a leading root-cause of crash-stop or timing failures within system components, and can be derived from either failure within a singular component or influence from other faulty and non-faulty component behaviour outside of its defined system boundaries. For example, a service that experiences high latency (due to stragglers or crash failures within the network) can result in communicating services to experience resource exhaustion. Performance interference between tasks in the same physical node results in performance degradation and resource exhaustion within other tasks. System designers attempt to mitigate such propagation via leveraging container-based mechanisms and cgroup restriction whose operation is dictated by quantitative QoS modelling to define the conservatively least resource boundary of each job group and fair resource allocations among different groups. However, determining the more 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 may be useful to enact automated decision making and alerting on-call technical staff, however encounters difficulties in terms of false negatives and false positives that may change in response to system usage. Therefore, a robust anomaly detection mechanism whose sensitivity can be appropriately tuned in accordance to current operational context of the system is required. A potential means to achieve this is leveraging adaptive learning of monitoring and detection parameter which considers different periodicity, parameter types, and parameter values. However how to generate and exploit streaming metrics to recognise outliers is intricately challenging due to the contradictory dilemma system monitor is facing - selective use of partial metrics to enact fast (yet imprecise) decisions or exploit a large amount of metrics for more precise (yet slow) decision making.

Preventive performance diagnosis. In contrast to reactive solutions whereby a faulty running service is halted to ascertain what conditions led to emergent failures manifestation to enact necessary maintenance (which has been demonstrated to be ineffective for dealing with stragglers [4]), a proactive diagnosis would ensure that end user service is minimally affected. Monitoring as many components as possible is likely to support failure prediction. However, in practise not all components can be monitored due to the sheer volume of data required to be collected, transmitted, and calculated. information pertaining to hardware to 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 due to the uncontrollable and intrinsic system factors. Statistical correlation among metrics can facilitate to rapidly to find root causes and determine 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. The performance refers to its ability to execute parallel applications and hold containerized services. Machine learning techniques such as classification and regression (e.g. Random Forest, Gradient Boosting Trees) might be one means in order 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, avoid assigning speculative tasks onto nodes that are likely to be in their weak performance state as soon as possible.

Cloud Scheduling and Instrumentation: Prevention and Tolerance

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

Latency-oriented tail mitigation based on redundancy. Modern cluster schedulers must deal with both the latency-sensitive requests and computation intensive tasks (e.g. long-running HTTP services, periodic *cron* jobs). Redundancy is the fundamental technique used to enhance component reliability of hardware, software and data storage. Based on the multi-replica 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 provisioned replica number at any given moment. Namely, the controller should launch a new replica if a component is killed or becomes inaccessible. For instance, in Kubernetes, the Replication Controller can auto-scale and manage microservices based on resource utilization or a fixed lower or upper limitations of expected replica number. For computation intensive tasks, the most common means to resolve the straggler is the speculation execution relying on the idempotency. However, a lack of coordinate fault tolerance between components leads to an emergent failure whereby such an action results in increased the resource contention, leading to cascading latencies for new tasks. Stragglers even rises more frequently in learning systems and distributed optimizations due to the performance is significantly throttled by slow communications and computations. The idempotency is invalid due to the shared states. Scenario-specific mitigations such as data encoding with built redundancy [15] and work reassignment with consistency bounds.

User-transparent failover and fault conversion. The system designer attempts to design the resource scheduler so that it can perform failover and self-healing (i.e. 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 within 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. While this strategy is an effective for recovering from incorrect state and data loss, the checkpointing itself is often considerably large. Checkpointing within a 1,000 node datacenter cluster in Alibaba over 24 hours has been reported to generate 1.7GB checkpoint (and within HPC, completing checkpointing can take hours to complete), and as demonstrated by the Amazon outage, can unknowingly manifesting 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 [13].However, due to the emergent failure cannot be anticipated beforehand, it is essential to enable the finite-state-machine (FSM) of system faults to more be able to adapt in accordance to detected system faults. For example, whether it is possible to automatically transform the emergent fault mode into a known classification of fault modes which can be then be tackled by established approaches. Once a fault is deterministic, the component or devices (such as storage block, NICs) 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 that could potentially be an effective approach for handling emergent failures. HFT relies on a cross-cutting components for system recovery tailored to a 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 HFT approach makes it possible to reduces the system complexity to addressing complex failure recovery scenarios.

For example, in order to address the challenges with performance interference, it could be possible to coordinate VMs on the same physical node. In the event where a VM is failing to adhere to timing requirements. The HFT could then consider performing coordinate recovery by leveraging components within both VMs. This could be facilitated by the hypervisor altering its scheduling to provide more CPU to a particular VM, and then measures the resultant delays in both to ensure satisfactory levels of CPU share. If they 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 is required that then incorporates the resource manager.

IoT integration. The presence of emergent failures is not solely confined to Cloud datacenters, and can manifest within any large-scale computing system. IoT is such a system particularly susceptible to emergent failures for many of the reasons given for Cloud – dynamic and unpredictable assortment of interconnected virtual and physical devices. A key difference is IoT exhibits high degree of dynamic join-leave not found within Cloud computing. If the system boundaries of interconnected components are constantly changing due to their usage and device composition, it is intuitive to assume that imposing rigid fault tolerance strategies that designed independently from the operational context of greater system is increasingly infeasible. Such system environments will also likely result in 'fluid' error confinement areas for a set of components, hence we believe a future research direction will be investigating how to autonomously determine the optimal fault tolerance and recovery mechanism for a given system context.

Conclusion

In this paper we discuss the rise of emergent failures: a growing problem towards 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 is (i) rethinking the nature of system abstraction allowing for holistic fault tolerance that cross-cuts coordination of components, and (ii) exploring the concept 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-grain analysis and observation, and towards creating models which precisely capture system conditions that lead to failure.

Acknowledgement

This work is supported by the EPSRC (EP/P031617/1) and the National Key Research and Development Program of China (2016YFB1000103). Dr. Renyu Yang is the corresponding author.

References

[1] Schroder, B., Gibson, A. G., A large-scale study of failures in high-performance computing systems, In the Proceedings of DSN, 2016.

[2] Garraghan, P., Solis Moreno, I. Townend, P., Xu, J. An Analysis of Failure-Related Energy Waste in a Large-Scale Cloud Environment, IEEE Transactions on Emerging Topics in Computing

[3] Garraghan, P., Ouyang, X., Yang, R., McKee, D., Xu, J., Straggler Root Cause and Impact Analysis for Massive scale Virtualized Cloud Datacenters, IEEE Transactions on Services Computing, 2016

[4] Dean, J., Barroso, L.A., The Tail at Scale, Communications of the ACM 56, 2013.

[5] Avižienis, A. et al., Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, 2004.

[6] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J., 2015, April. Large-scale cluster management at Google with Borg. In Proceedings of the Tenth European Conference on Computer Systems (p. 18). ACM.

[7] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J., 2016. Borg, omega, and kubernetes. ACM Queue, 14(1), p.10.

[8] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R.H., Shenker, S. and Stoica, I., 2011, March. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In NSDI (Vol. 11, No. 2011, pp. 22-22).

[9] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S. and Saha, B., 2013, October. Apache hadoop yarn: Yet another resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (p. 5). ACM.

[10] Zhang, Z., Li, C., Tao, Y., Yang, R., Tang, H. and Xu, J., 2014. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale. Proceedings of the VLDB Endowment, 7(13), pp.1393-1404.

[11] Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J., Qian, Z., Wu, M. and Zhou, L., 2014, October. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In OSDI (Vol. 14, pp. 285-300).

[12] Maurer, B., 2015. Fail at scale. ACM Queue, 13(8), p.30.

[13] Yang, R., Zhang, Y., Garraghan, P., Feng, Y., Ouyang, J., Xu, J., Zhang, Z. and Li, C., 2017. Reliable computing service in massive-scale systems through rapid low-cost failover. IEEE Transactions on Services Computing, 10(6), pp.969-983.

[14] Sun, X., Hu, C., Yang, R., Garraghan, P., Wo, T., Xu, J., Zhu, J. and Li, C. ROSE: Cluster Resource Scheduling via Speculative Over-subscription. In IEEE ICDCS 2018

[15] Karakus, C., Sun, Y., Diggavi,S. and Yin, W., 2017. Straggler mitigation in distributed optimization through data encoding. In Advances in Neural Information Processing Systems (pp. 5434-5442).

ABOUT THE AUTHORS

Peter Garraghan is a Lecturer in Distributed Systems at Lancaster University, UK, and received his PhD in Computer Science at the University of Leeds, UK. His research interests encompass massive-scale distributed systems, dependability, resource management, and energy-efficiency. Contact him at p.garraghan@lancaster.ac.uk

Renyu Yang is a research fellow at University of Leeds, UK and R&D scientist at Edgetic Ltd., UK. His research interests include massive-scale distributed systems, resource scheduling, and dependability. He received his PhD in Computer Science from Beihang University, China. Contact him at r.yang1@leeds.ac.uk

Zhenyu Wen is research fellow at Newcastle University, UK. He received his PhD in Cloud Computing from Newcastle University. His research interests includes IoT, Distributed systems, Big Data Analytics and Computer network. Contact him at zhenyu.wen@newcastle.ac.uk.

Alexander Romanovsky is a Chair Professor of Computing Science at Newcastle University, UK. He received his PhD in Computer Science from St. Petersburg State Technical University, Russia. His research interests include fault tolerance and system dependability. Contact him at alexander.romanovsky@ncl.ac.uk.

Jie Xu is a Chair Professor of Computing at the University of Leeds, Lead of a Research Peak of Excellence at Leeds, and Head of Distributed Systems and Services, and co-founder of Edgetic Ltd. UK. His research interests include large-scale distributed computing and dependability etc. Contact him at j.xu@leeds.ac.uk

Rajkumar Buyya is a professor of computer science and software engineering, at the University of Melbourne, Australia. His research interests include cloud, grid, distributed, and parallel computing. Buyya has 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 IoT at Newcastle University, UK. He received a PhD in Computer Science and Software Engineering from the University of Melbourne. His research interests include Internet of Things, Big Data Analytics. Contact him at raj.ranjan@ncl.ac.uk.