The Case for Determinism on the Edge

Matthew Furlong
University of Michigan

Andrew Quinn
University of Michigan

Jason Flinn
University of Michigan

Abstract

Emerging edge applications, such as augmented and virtual reality, real-time video analytics and thin-client gaming, are latency-sensitive, resource-intensive, and stateful. Transitioning these applications from cloud deployments to the edge is non-trivial since edge deployments will exhibit variable resource availability, significant user mobility, and high potential for faults and application preemption, requiring considerable developer effort per application to maintain stable quality of experience for the user.

In this paper, we propose *deterministic containers*, a new abstraction that simplifies the development of complex applications on the edge. Deterministic containers enforce the property that all activity within a container behave deterministically. Determinism provides replication, which in turn provides key benefits for edge computing including resilience to performance jitter, enhanced fault-tolerance, seamless migration, and data provenance.

We are currently building a prototype, Shadow, that aims to provide deterministic containers with minimal performance overhead while requiring few application modifications. For all sources of non-determinism, Shadow either converts the behavior to be deterministic or restricts the allowable application behavior. Preliminary results indicate that using Shadow to reduce performance jitter at the edge for a vehicle caravan application involving video analytics reduces median application response time by up to 25%.

1 Introduction

Many emerging edge applications are latency-sensitive, resource-intensive, and stateful. For example, augmented and virtual reality, real-time video analytics, and thin-client gaming all share these properties. Currently proposed edge infrastructure is ill-suited to support these types of applications since typical edge deployments exhibit highly variable resource availability, significant user mobility, and a much higher potential for faults and application preemptions than seen in controlled data center environments. In addition, the integration of a myriad of potentially-sensitive data sources with computation running in heterogeneous edge infrastructure, e.g., for emerging IoT applications, creates the need to track and potentially act on detailed data provenance.

This observation poses a challenge to developers who wish to convert their complex, resource-intensive applications from running in the data center to running on edge infrastructure. Achieving equivalent reliability, mitigating performance variation, and adding new features such as detailed provenance tracking seemingly each require substantial application modification, requiring considerable developer effort to move from the cloud to the edge.

In this paper, we observe that a new type of system support in edge infrastructure would make moving complex applications from the cloud to the edge much easier. This abstraction, *deterministic containers*, enforces the property that all activity within a container behaves *deterministically*, i.e., that each execution of the application will reach identical program states and produce identical outputs when provided the same inputs. Note that most complex applications are inherently non-deterministic and thus are not guaranteed to reach the same state or produce the same output even when run with the same inputs.

Determinism is important because it is a foundational requirement for state-machine replication: two replicas will reach the same state and produce the same output if provided the same input. Replication in turn can provide many benefits for edge computing: for example, performance jitter can be reduced by executing multiple replicas on different edge servers and using the fastest result, reliability can be enhanced by running a delayed backup in the cloud or using standard state machine replication across edge nodes, applications can be seamlessly migrated across edge nodes without downtime, and data provenance can be tracked efficiently by recording only the inputs to the container and using a future replicated execution to extract provenance information such as input/output causality.

Services in the cloud traditionally use a replicated state

store (e.g., Cassandra) to ensure consistency across replicas. However, emerging edge applications have large internal state that is constantly being updated; replicating this state using strong consistency would limit the benefits of edge computing due to the overhead of consistency protocols, while eventual consistency could cause the output from non-deterministic replicas to diverge in meaningful ways (e.g., replicas of a parking assistant may choose to park in two different parking spots). Deterministic containers instead provide a lightweight replication solution better suited for emerging edge applications.

In this paper, we discuss many potential such use cases for deterministic containers at the edge. Using deterministic containers, applications automatically realize these complex properties with minimal modifications.

We are building a prototype, named Shadow, that aims to provide deterministic containers with minimal performance overhead while requiring few application modifications. Our approach to developing Shadow is to tackle each major source of non-determinism in turn. We either convert inherently non-deterministic behavior to be deterministic or restrict the allowable application behavior such that any execution within the container is deterministic. Thus, if an application runs within the container, it behaves deterministically; if it cannot run as-is within the container, developers must modify it to fit into a restricted execution model (in which case, our hypothesis is that such modifications are much easier than, e.g., adding fault-tolerance, jitter resistance, and provenance tracking).

We have examined several major sources of non-determinism so far. First, Shadow uses a deterministic thread scheduler based on Kendo [29] to enforce consistent thread ordering. This approach provides low-overhead determinism, but restricts support to only race-free applications. Second, Shadow hooks non-deterministic system calls such as time and random numbers to provide deterministic results. Third, Shadow uses proxy-agents on the client and server to provide deterministic network communication between these entities without requiring modifications to the application. Finally, Shadow provides deterministic third-party services by using the client proxy-agent to manage all third-party requests.

In the rest of the paper, we first outline the potential use cases for deterministic containers at the edge. We next discuss each of the major sources of non-determinism in turn and how Shadow can convert non-deterministic behavior to deterministic execution for each one. Finally, we show preliminary results using Shadow to reduce performance jitter at the edge for a vehicle caravan application involving video analytics, in which median application response time is improved by up to 25%.

2 Use Cases

In this section, we discuss a few of the potential use cases for deterministic containers at the edge.

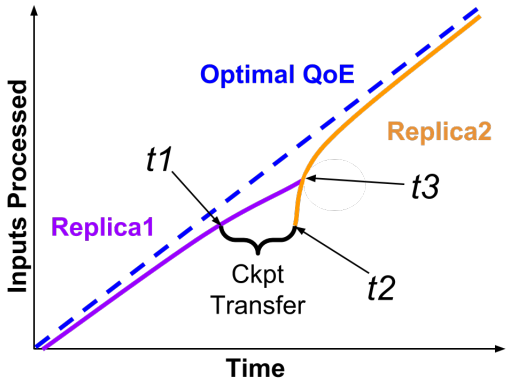

Figure 1: Migration Through Replication: As the client moves away from Replica1, the application performance deteriorates. At t_1, the application triggers migration by cloning Replica1 and migrating the container to a closer edge server. At time t_2, Replica2 has finished migration so the application forwards all inputs from after t_1 to Replica2. At t_3, Replica2 reaches an identical state to Replica1, finishing migration.

2.1 Reducing application response time

To provide high quality of experience (QoE) for the user, many emerging edge applications require reliable, low-latency edge services [14, 19]. Unfortunately, studies suggest that edge resources will be limited [18] and networks are known to have significant variability for mobile users [21]. With many applications competing for limited resources, tail latency will be an unfortunate certainty for edge computing, which can lead to unpredictable delays and uneven QoE.

Deterministic containers can mask the inherent latency variability in edge computing caused by unpredictable wireless networks, mobility, and uneven server provisioning through replication. An application running in a deterministic container can be trivially replicated at multiple edge nodes connected by different wireless networks. The client sends each service request to all replicas and uses the response from the replica that responds first. Determinism guarantees that all other replicas will produce the same output. Further, they will arrive at the same application state after processing the request, so the application state will not diverge. This approach is commonly used to mask latency variability in the data center for deterministic applications [10]; deterministic containers allow the same technique to be applied to non-deterministic applications at the edge.

Our prototype currently supports this use case. We later show that deterministic containers reduce median response time by up to 25% and tail 99% response time by 17.5% (see Section 4).

2.2 Migration

In order to provide an immersive user experience, emerging edge applications attempt to maintain low response times

by offloading their computation to the ”closest” edge server (i.e., the server with lowest network latency). Prior systems propose service handoff [8, 16, 24, 25, 33], in which the server component of an application migrates from one edge node to another. The cost of service handoff is measured by both the total service migration time and service downtime (i.e., the amount of the time the service is suspended while it is moved).

Prior systems for service handoff typically use pre-copy [9], which iteratively migrates dirty pages from the application until the dirty set reaches a predefined size and then suspends and transfers the remaining dirty pages. Depending on the selected size, pre-copy favors either reduced downtime or reduced migration time. Systems favoring downtime at the cost of completion time degrade the user's QoE as the client must continue using the original edge server past when the service would ideally have been migrated. If the system instead minimizes completion time, the application will experience lengthy periods of unavailability.

With deterministic containers, downtime can be minimized by using replication rather than migration to move stateful services to a new edge node. First, using copy-on-write, the container clones itself. Next, the cloned container is migrated to the destination edge server while the original container continues processing application inputs. Then, the client application supplies all inputs from the time of migration to the new container. Since execution is deterministic, the migrated container is guaranteed to eventually reach an identical state as the original application, at which point the migration is complete. Figure 1 shows the migration via replication approach.

2.3 Fault Tolerance

To maintain high availability, edge applications must be resilient to infrastructure failure. Compared to the data center, failures are likely to be more common at the edge because edge applications must rely on heterogeneous services from numerous providers, edge infrastructure may operate in distributed and less controlled environments, and many applications may compete for limited edge resources against high-priority services (e.g., 5G), leading to unexpected preemptions.

In the data center, state machine replication provides fault tolerance. However, the applications that use replication are carefully designed to be deterministic. In contrast, deterministic containers allow stateful edge applications to use state machine replication across multiple edge nodes for reliability, even for applications not inherently designed to be fault tolerant. The fault tolerance layer, e.g., a Paxos or distributed shared log implementation, can be written in an generic fashion and simply invoke any application operation, knowing that it will be deterministic.

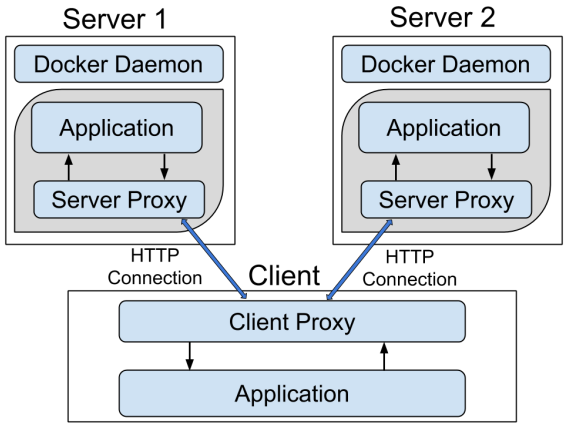

Figure 2: Overall Design of Shadow

2.4 Data Provenance

The explosive growth of Internet of Things (IoT) devices has led to increasing collection of sensitive user data, requiring new abstractions for privacy and trust. Additionally, edge deployments leverage services from many providers such as internet service providers, cloud service providers, or a local private cloud [34]. In these complex, multi-party scenarios, differentiating between correct and malicious use of data requires tracking and auditing data provenance, as well as low-level behavior of the application [15].

Repeatability is the ultimate form of provenance. If one can recreate a past computation, then the recreated execution can be instrumented and examined to answer any question of how inputs were consumed or how outputs were generated. Determinism makes such recreation much easier: if the system preserves the input to the computation or retains the ability to recreate such inputs (e.g., by re-executing a previous computation in a pipeline), then it can create a replica that repays the past execution on demand. That replica can be instrumented and used to answer any provenance query. Using determinism, auditing can be performed in parallel to the application on an external machine [27, 28] or performed retroactively [13].

3 Design

Figure 2 shows the high-level design of Shadow, our prototype for deterministic containers. Shadow supports both simultaneous deployments of deterministic containers for replication and recording inputs for future executions (i.e., record-and-replay scenarios). Shadow is designed for deployments where the server component of the application serves a single client. For each source of non-determinism, Shadow either converts non-deterministic application behavior to be deterministic or restricts allowable application behavior so that any execution within the container is deterministic. In this section, we describe how Shadow can address several major sources of

non-determinism.

3.1 Multiple threads

When an application shares memory across multiple threads, non-deterministic states arises because the order of accesses to shared memory are unconstrained. To eliminate this source of non-determinism, Shadow must enforce a deterministic order for shared memory accesses.

Unfortunately, techniques that deterministically order *all* shared memory accesses are too slow to be practical [3], so Shadow instead enforces weak determinism [29], in which a low-overhead thread scheduler deterministically order all synchronization operations within an application. This guarantees that any race-free application behaves deterministically, but provides no guarantee for racy programs. Thus, Shadow requires that applications executing in the container be data race free. There are many popular tools for identifying data races [32], so prior work has shown that many applications are race-free [11, 26].

3.2 Client Inputs

Deterministic replication requires that each container receive the same sequence of inputs from a client. Further, this input must be received at the same point in the application execution for all replicas. Shadow therefore multicasts client input to all containers replicating a given service. To support offline replication, e.g., provenance tracking, Shadow records the input and the point in the execution at which it was delivered; it supplies the same input at the same point for any subsequent execution.

Shadow avoids modifying applications by using proxy agents on both the client and the server for communication, as shown in Figure 2. The client agent multicasts requests to all server proxies, applying sequence numbers to order the requests. It collects responses from multiple replicas, but supplies only a single output (from the replica that responds the quickest). The proxy agents communicate using stateless HTTP connections, which simplifies the cloning of deterministic containers (see Section 2.2).

To deliver input at the same point in application execution, Shadow restricts the points at which inputs can be delivered. Currently, it executes the containerized application in *epochs*, in which a set number of input bytes are delivered at the beginning of each epoch; Shadow blocks execution at the start of the epoch until it can deliver the set number of bytes. Inputs that arrive in the middle of an epoch are buffered until the start of the next epoch.

3.3 Operating System Events

The operating system interface is a major source of non-determinism. The operating system exposes many non-deterministic values via system calls such as `gettimeofday` and `getrandom`), and it provides non-deterministic channels for inter-process communication and reading from kernel devices.

Where feasible, Shadow converts non-deterministic system calls to be deterministic. Container namespaces provide determinism in the application environment (e.g., `getpid`). For random numbers, the client proxy passes a random number seed alongside each user input, following the approach advocated by PBFT [7]. The container reseeds a psuedo-random number generator at the start of processing for each epoch.

Providing deterministic time is challenging since time must monotonically increase during execution. The client proxy sends a timestamp with each client request that is used to seed the time at the start of the epoch. Shadow shims each system call made by the process group to increment this time value by a constant delta per system call executed. This provides deterministic execution, but may also skew time measurements. Applications that require fine-grained real time measurements may not run correctly within a deterministic container.

Most file system operations are already deterministic, except for their handling of timestamps which are handled using the same timing mechanism as above.

Known techniques to make inter-process communication deterministic are too expensive to use in edge scenarios [4], so Shadow currently disallows inter-process communication within the container. It also does not support applications that directly read kernel devices. We plan to gradually relax our restrictions as our prototype is developed.

3.4 External Sources of Non-Determinism

Finally, non-determinism can arise from communication with external services.

Shadow provides determinism by using the client proxy agent to mediate all communication with third-parties. The server proxy forwards requests to external services to the client agent (currently, this may require application modification). When the client proxy receives a new third-party request (i.e., no containers have made this particular request yet), it contacts the third party, stores the output, and returns the result to the server. If the client determines that the third-party request was already issued by another container, the client agent returns the stored result to the server. Since we expect low latency between the client and server in edge deployments, the additional performance overhead of the proxy indirection should be minimal.

4 Results

To explore the benefits of deterministic containers, we measured our Shadow prototype using deterministic containers to mask latency variation. In each container, we run a vehicle caravan application that tracks a vehicle in the caravan that the driver is following. The application is designed to high-

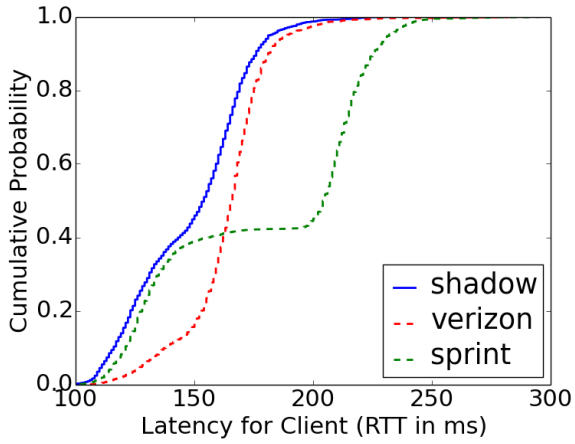

Figure 3: CDF of application response time for a vehicle caravan application. For Sprint and Verizon, we use non-deterministic containers and emulate network latency using Sprint and Verizon traces. For Shadow, we replicate using two deterministic containers, with the Verizon trace to emulate network latency for the first replica and the Sprint trace to emulate network latency for the second.

light the tracked vehicle on the windshield using a heads-up display. The application uses OpenCV [5] for object recognition and tracking. We emulate network conditions using traces of a vehicle driving in downtown Ann Arbor collected from VNperf [21]. The traces contain latency measurements simultaneously gathered from two different networks, Verizon and Sprint, and represents latencies to city-scale edge infrastructure potentially located within these networks.

We evaluate the quality of experience of the caravan application using the client perceived request latency for each video frame. We evaluate the performance of offloading without replication by deploying the application in a container with determinism disabled and emulating the network latency from either the Verizon or Sprint trace. To evaluate Shadow, we enable determinism and deploy two replicated deterministic containers. We use the Verizon trace to emulate network latency for the first replica and the Sprint trace to emulate network latency for the second.

Our results in Figure 3 show that Shadow significantly improves QoE for the caravan application by using the response from the container that replies first. Interestingly, Shadow not only decreases tail latency; it also consistently outperforms both unreplicated tests. On average, Shadow's replicated approach is 9.7% faster than Verizon and 17.4% faster than Sprint. At the 99th percentile, Shadow is 9.9% faster than Verizon and 17.7% faster than Sprint.

5 Related Work

Our work is the first to propose deploying deterministic containers at the edge.

Prior systems to enforce determinism for processes have high overhead and do not eliminate external sources of non-determinism [4]. Deterministic thread schedulers have shown more promising performance, and Shadow's deterministic thread scheduler is based on the weak determinism introduced by Kendo [29]. Weak determinism provides determinism with low-overhead for race-free applications. Other deterministic thread schedulers are a poor fit for Shadow due to high performance overhead [3]. Dthreads [23] are lightweight, but require an individual process for each thread, making them a poor fit for deterministic container use cases such as migration.

State machine replication traditionally assumes the underlying application is deterministic in order to provide consistency. Eve [20] allows replication of non-deterministic applications by using an execute-then-verify approach, which validates that all replicas have reached the same state. Eve requires a primary node for the verify step, making it a poor fit for using redundancy to reduce application response time on the edge. Instead, deterministic containers allow replication of applications that are inherently non-deterministic without requiring the replicas to coordinate, alleviating these limitations.

Application migration has been extensively studied in the context of processes [12, 22, 31], containers [24, 30] and virtual machines [9, 16]. These systems encounter a trade-off between service downtime and migration time. Deterministic containers avoid this trade-off by using replication rather than migration to move stateful services to new edge nodes.

Existing edge deployment solutions (e.g., Amazon Greengrass [1], Microsoft IoT Edge [2]) require developers to build stateless services in order to realize similar benefits to deterministic containers. Emerging applications, such as real-time video analytics and augmented reality, are inherently stateful. Porting these applications into a stateless model will require using a replicated storage service which can lead to high overhead [17].

Prior systems have also shown that determinism aids with fault tolerance [6] and data provenance [11, 13]. We build on this work.

6 Conclusion

In this paper, we propose deterministic containers to ease the transition of applications from data centers to the edge. Deterministic containers facilitate replication, which enables seamless migration, resilience to performance jitter, fault tolerance and data provenance. Our prototype, Shadow, provides low-overhead deterministic containers by converting non-deterministic behavior to become deterministic or by placing restrictions on the possible application executions. Preliminary results show that Shadow improves median response time of a video analytics application by up to 25% in the presence of network jitter.

References

[1] Aws iot greengrass: Bring local compute, messaging, data caching, sync, and ml inference capabilities to edge devices, 2018.

[2] Azure iot edge: Extend cloud intelligence and analytics to edge devices, 2018.

[3] BERGAN, T., ANDERSON, O., DEVIETTI, J., CEZE, L., AND GROSSMAN, D. Coredet: A compiler and runtime system for deterministic multithreaded execution. *SIGARCH Comput. Archit. News 38*, 1 (Mar. 2010), 53–64.

[4] BERGAN, T., HUNT, N., CEZE, L., AND GRIBBLE, S. D. Deterministic process groups in dos. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 177–191.

[5] BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[6] BRESSOUD, T. C., AND SCHNEIDER, F. B. Hypervisor-based fault tolerance. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1995), SOSP '95, ACM, pp. 1–11.

[7] CASTRO, M., AND LISKOV, B. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 1999), OSDI '99, USENIX Association, pp. 173–186.

[8] CHAUFOURNIER, L., SHARMA, P., LE, F., NAHUM, E., SHENOY, P., AND TOWSLEY, D. Fast transparent virtual machine migration in distributed edge clouds. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (New York, NY, USA, 2017), SEC '17, ACM, pp. 10:1–10:13.

[9] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation-Volume 2* (2005), USENIX Association, pp. 273–286.

[10] DEAN, J., AND BARROSO, L. A. The tail at scale. *Communications of the ACM 56* (2013), 74–80.

[11] DEVECSERY, D., CHOW, M., DOU, X., FLINN, J., AND CHEN, P. M. Eidetic systems. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation* (2014), pp. 525–540.

[12] DOUGLIS, F., AND OUSTERHOUT, J. Transparent process migration: Design alternatives and the sprite implementation. *Softw. Pract. Exper. 21*, 8 (July 1991), 757–785.

[13] DUNLAP, G. W., KING, S. T., CINAR, S., BASRAI, M. A., AND CHEN, P. M. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th symposium on Operating systems design and implementation* (2002), USENIX Association, pp. 211–224.

[14] ELLIS, S. R., MANIA, K., ADELSTEIN, B. D., AND HILL, M. I. Generalizeability of latency detection in a variety of virtual environments. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (2004), pp. 2632–2636.

[15] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 393–407.

[16] HA, K., ABE, Y., EISZLER, T., CHEN, Z., HU, W., AMOS, B., UPADHYAYA, R., PILLAI, P., AND SATYANARAYANAN, M. You can teach elephants to dance: Agile vm handoff for edge computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)* (2017).

[17] HELLERSTEIN, J. M., FALEIRO, J., GONZALEZ, J. E., SCHLEIER-SMITH, J., SREEKANTI, V., TUMANOV, A., AND WU, C. Serverless computing: One step forward, two steps back, 2018.

[18] HONG, C.-H., AND VARGHESE, B. Resource management in fog/edge computing: A survey.

[19] ITU-REPORT. The tactile internet, Aug 2014.

[20] KAPRITSOS, M., WANG, Y., QUEMA, V., CLEMENT, A., ALVISI, L., AND DAHLIN, M. All about eve: Execute-verify replication for multi-core servers. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 237–250.

[21] LEE, H., FLINN, J., AND TONSHAL, B. Raven: Improving interactive latency for the connected car. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (New York, NY, USA, 2018), MobiCom '18, ACM, pp. 557–572.

[22] LITZKOW, M., TANNENBAUM, T., BASNEY, J., AND LIVNY, M. Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System. Technical Report CS-TR-1997-1346, University of Wisconsin, Madison, Apr. 1997.

[23] LIU, T., CURTSINGER, C., AND BERGER, E. D. Dthreads: Efficient deterministic multithreading. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles* (2011), SOSP '11, ACM.

[24] MA, L., YI, S., AND LI, Q. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC)* (2017).

[25] MACHEN, A., WANG, S., LEUNG, K. K., KO, B. J., AND SALONIDIS, T. Live service migration in mobile edge clouds. *Wireless Commun. 25*, 1 (Feb. 2018), 140–147.

[26] MASHTIZADEH, A. J., GARFINKEL, T., TEREI, D., MAZIERES, D., AND ROSENBLUM, M. Towards practical default-on multi-core record/replay. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2017), ASPLOS '17, ACM, pp. 693–708.

[27] MING, J., WU, D., XIAO, G., WANG, J., AND LIU, P. Taintpipe: pipelined symbolic taint analysis. In *24th {USENIX} Security Symposium ({USENIX} Security 15)* (2015), pp. 65–80.

[28] NIGHTINGALE, E. B., PEEK, D., CHEN, P. M., AND FLINN, J. Parallelizing security checks on commodity hardware. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2008), ASPLOS XIII, ACM, pp. 308–318.

[29] OLSZEWSKI, M., ANSEL, J., AND AMARASINGHE, S. Kendo: Efficient deterministic multithreading in software. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2009), ASPLOS XIV, ACM, pp. 97–108.

[30] OSMAN, S., SUBHRAVETI, D., SU, G., AND NIEH, J. The design and implementation of zap: A system for migrating computing environments. In *Proceedings of the 5th Symposium on Operating Systems Design and implementationCopyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading* (Berkeley, CA, USA, 2002), OSDI '02, USENIX Association, pp. 361–376.

[31] POWELL, M. L., AND MILLER, B. P. Process migration in demos/mp. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1983), SOSP '83, ACM, pp. 110–119.

[32] SEREBRYANY, K., AND ISKHODZHANOV, T. Threadsanitizer: data race detection in practice. In *Proceedings of the workshop on binary instrumentation and applications* (2009), ACM, pp. 62–71.

[33] TEKA, F., LUNG, C.-H., AND AJILA, S. A. Nearby live virtual machine migration using cloudlets and multipath tcp. *J. Cloud Comput. 5*, 1 (Dec. 2016), 61:1–61:21.

[34] YI, S., QIN, Z., AND LI, Q. Security and privacy issues of fog computing: A survey. In *WASA* (2015).