Exploiting content similarity to address cold start in container deployments

Kunal Mahajan
Columbia University

Saket Mahajan
Santa Clara University

Vishal Misra
Columbia University

Dan Rubenstein
Columbia University

ABSTRACT

Serverless computing is an emerging Cloud paradigm that allows users to claim and pay for resources only when their jobs are executing. While this paradigm offers several advantages, the phenomenon of "cold start" reduces its inherent efficiency with respect to the utilization of compute, storage and network resources that support its existing virtualization deployment systems. We analyze current modes of deployment and identify data similarities across applications. Based on these observations, we propose a new deployment system that is built atop a peer-to-peer network, virtual file-system and content-addressable storage, which will increase compute availability, reduce storage requirement, and prevent network bottlenecks.

CCS CONCEPTS

• **Information systems** → **Distributed storage**; • **Computer systems organization** → **Cloud computing**; **Peer-to-peer architectures**; • **Networks** → *Network experimentation*; • **Software and its engineering** → *File systems management*;

ACM Reference Format:
Kunal Mahajan, Saket Mahajan, Vishal Misra, and Dan Rubenstein. 2019. Exploiting content similarity to address cold start in container deployments. In *The 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19 Companion), December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3360468.3366781

1 INTRODUCTION

CISCO[8] has forecasted that 94 percent of workloads and compute instances will be processed by cloud data centers and the annual global data center IP traffic will reach 20.6 ZB by the end of 2021 [21]. This begs the question: *Is the current cloud system capable of efficiently utilizing compute, storage and network resources for the increased workload?*

Serverless computing has emerged as a win-win technology benefiting both cloud providers (CP) by providing fine-grained control and the users by allowing quick application deployments, version control, and attractive pricing [1, 5, 7, 10, 22–27, 29, 30, 33, 34]. In serverless, the CP maintains generic worker nodes to execute the functions uploaded by the users from a centralized *storage bucket* (e.g., S3[2]) that holds code to be executed when corresponding jobs arrive. For function execution, the CP builds a container with the source code, downloads the container on a worker node from centralized storage bucket and executes it. This process induces a latency while downloading and initializing the container, referred to as a "cold start", whose magnitude is dependent on a set of application-specific factors [18, 29, 32]. The cold start problem is exacerbated with scaling as each new function execution incurs the cold start latency. To reduce the impact of cold start for subsequent executions, multiple instances of the worker node are retained for tens of minutes [32], thereby reducing compute availability.

The cold start problem arises due to the existing container deployment design, which retrieves each new instance of a container from the storage bucket. Our proposed system design addresses the problem and consists of 4 modules: peer-to-peer network (p2p), virtual file system with content-addressable storage (CAS), partial delivery execution and TCP splitting. The user uploads the source code to a CAS bucket that has hash based references to file blocks using a MerkleDAG [28]. Each worker node runs a p2p client and a CAS-based virtual file system. To execute the containerized source code, the worker node requests file blocks from the virtual file system, which serves the blocks if already present at the node; otherwise, it fetches the requested file blocks from the p2p network from other nodes and/or the CAS bucket. This enables partial delivery execution[31] (starting execution before all blocks are delivered) of the container, with the added benefit of live migration [9] of the container from one worker node to another. Live migration requires availability of the application's IP address on the destination machine, which we ensure via TCP splitting at the master node as shown in figure 1. Our system can be incorporated in existing container orchestration platforms such as Kubernetes [17].

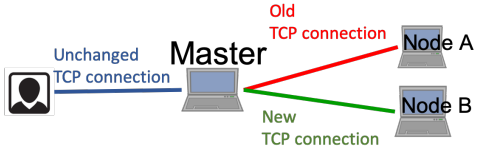

Figure 1: TCP splitting

This work was funded in part by the NSF through awards CNS-1717867, CNS-1618911 and CNS-1910138. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CoNEXT '19 Companion, December 9–12, 2019, Orlando, FL, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7006-6/19/12…$15.00
https://doi.org/10.1145/3360468.3366781

2 KEY IDEAS

The key ideas are based on analysis of the characteristics of the different ways the applications are deployed, and motivating the need for hash-based reference to file blocks.

2.1 Characteristics of deploying an application

Scaling [11], versioning [6], and live migration [9] represent the three ways in which an application is deployed on the worker nodes. In scaling, multiple identical application instances are replicated across nodes to meet either the current demand or predicted demand. In versioning, the application is updated, requiring different versions to be supported on the cloud. The updated application will likely have significant overlaps of data with previous versions. In live migration, an application is moved from one worker node to another, which requires saving current state (checkpoint) of the container, transferring and executing the checkpoint on a different worker without disrupting the user experience. For all the three modes, our proposed system can reduce the latency by exploiting the presence of identical blocks. Figure 2 shows the percentage of identical $256KB$ blocks of 15 checkpoints of an open source e-commerce web application, PrestaShop [13], taken after every minute. As shown in figure, the first checkpoint had 53% blocks identical to the original container checkpoint. Even in later checkpoints there still remained a 24% overlap with the original configuration of the application.

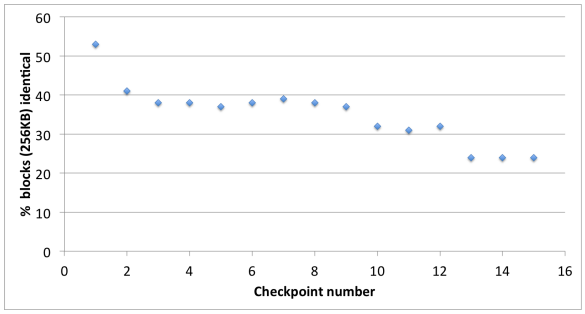

Figure 2: Blocks similarity

2.2 Block similarity in source code across applications

Applications tend to commonly use popular libraries for a variety of tasks, such as image processing, data analytics, machine learning, etc. Based on scraping 876K Python projects from GitHub, [29] identified 20 most common PyPI[14] packages used as GitHub project dependencies and measured the initialization time for these packages with 12.8 seconds as the highest time for pandas[12]. This time can be reduced by our proposed system as the blocks for the popular libraries may already exist on the worker node and/or can be fetched from multiple worker nodes by leveraging the p2p network.

3 IMPLEMENTATION AND EVALUATION

Our system utilizes modified IPFS [19], a p2p, content addressed file system. The IPNS [19] mount-point provides the virtual FUSE-based custom file system enabling container executions and live migrations. The evaluation below shows twofold benefits: minimizing average completion time of container checkpoint transfers and reduction in the cold start latency.

3.1 Multiple Shuffles

Our testbed consists of 8 t2.micro[16] nodes on AWS[4], four in Oregon region and the remaining in North Virginia region. Each node has 2 container images (roughly 200MB each). In shuffle, each node acquires 2 container images chosen at random from nodes in the other region. We perform multiple shuffles capturing scenarios in big data distributed processing frameworks such as Apache Spark [3]. Figure 3 shows an improvement of 33.3% and 59.3% in average transfer completion time over rsync[15] transfers for the first shuffle and sixth shuffle respectively.

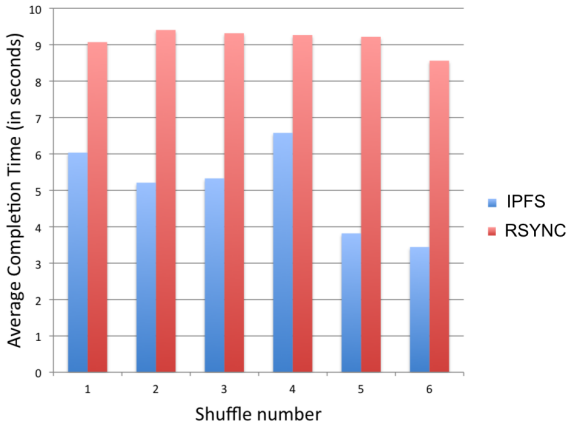

Figure 3: Multiple shuffles

3.2 Live Migration

We developed a simple web application which simply kept a count of number of HTTP requests and created a checkpoint whose size was 7MB. We performed live migration of the container using our system and found a 37.9% reduction in boot time over rsync on the same testbed as in sec 3.1 with 2 nodes containing copies of the checkpoint.

4 RELATED WORK

[29] proposed a specialized container system, eliminating kernel bottlenecks, that is optimized for serverless workloads. Other solutions to cold start problem include increasing the memory allocation, choosing a faster runtime, keeping shared data in memory, shrinking source code package size, monitoring performance and logging relevant indicators, using time-series forecasting, and keeping a pool of pre-warmed functions [20]. However these solutions not only put an additional burden on the developers by increasing monetary costs, reducing implementation flexibility, requiring significant optimization, thereby hindering quick deployment and fast growth, but also inefficiently utilize the compute and storage resources of the cloud provider.

REFERENCES

[1] 2017. Apache Openwhisk: https://github.com/apache/incubator-openwhisk. (2017). https://github.com/apache/incubator-openwhisk
[2] 2019. Amazon S3. https://aws.amazon.com/s3/. (2019).
[3] 2019. Apache Spark. https://spark.apache.org/. (2019).
[4] 2019. AWS. https://aws.amazon.com. (2019).
[5] 2019. AWS Lambda. https://aws.amazon.com/lambda/. (2019).
[6] 2019. AWS Lambda Function Versioning and Aliases. https://docs.aws.amazon.com/lambda/latest/dg/versioning-aliases.html. (2019).
[7] 2019. Azure Functions. https://azure.microsoft.com/en-us/services/functions/. (2019).
[8] 2019. CISCO. https://www.cisco.com/. (2019).
[9] 2019. CRIU live migration. https://criu.org/Live_migration. (2019).
[10] 2019. Google Cloud Functions. https://cloud.google.com/functions/. (2019).
[11] 2019. Kubernetes-Engine: Scaling an application. https://cloud.google.com/kubernetes-engine/docs/how-to/scaling-apps. (2019).
[12] 2019. Pandas: Python Data Analysis Library. https://pandas.pydata.org/. (2019).
[13] 2019. PrestaShop. https://github.com/PrestaShop/docker. (2019).
[14] 2019. Python Package Index. https://pypi.org/. (2019).
[15] 2019. Rsync. https://linux.die.net/man/1/rsync. (2019).
[16] 2019. T2 micro instance. https://aws.amazon.com/ec2/instance-types/t2/. (2019).
[17] The Kubernetes Authors. 2019. Kubernetes. https://kubernetes.io/. (2019).
[18] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. *Serverless Computing: Current Trends and Open Problems.*
[19] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* (2014).
[20] Renato Byrro. 2019. Can we solve serverless cold starts? https://dashbird.io/blog/can-we-solve-serverless-cold-starts/. (2019).
[21] Cisco. 2018. CISCO white paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html. (2018).
[22] Matt Crane and Jimmy Lin. 2017. An Exploration of Serverless Architectures for Information Retrieval. In *ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR).*
[23] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. Serverless Computation with OpenLambda. In *USENIX Workshop on Hot Topics in Cloud Computing.*
[24] Qingye Jiang, Young Choon Lee, and Albert Y. Zomaya. 2017. Serverless Execution of Scientific Workflows. In *Service-Oriented Computing.*
[25] Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *ACM Symposium on Cloud Computing (SoCC).*
[26] Kunal Mahajan, Daniel R Figueiredo, Vishal Misra, and Dan Rubenstein. In Press. Optimal Pricing for Serverless Computing. In *IEEE Global Communications Conference (Globecom 2019).*
[27] G. McGrath and P. R. Brenner. 2017. Serverless Computing: Design, Implementation, and Performance. In *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW).*
[28] Ralph Merkle. 1987. A Digital Signature Based on a Conventional Encryption Function. *LNCS.*
[29] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *USENIX Annual Technical Conference (ATC).*
[30] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI).*
[31] Joshua Reich, Oren Laadan, Eli Brosh, Alex Sherman, Vishal Misra, Jason Nieh, and Dan Rubenstein. 2012. VMTorrent: scalable P2P virtual machine streaming. In *CoNEXT.*
[32] Mikhail Shilkov. 2019. Cold Starts in AWS Lambda. https://mikhail.io/serverless/coldstarts/aws/. (2019).
[33] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. 2016. Building a Chatbot with Serverless Computing. In *ACM International Workshop on Mashups of Things and APIs (MOTA).*
[34] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *USENIX Annual Technical Conference (ATC).*