Pocket

Elastic Ephemeral Storage for Serverless Analytics

ANA KLIMOVIC, YAWEN WANG, PATRICK STUEDI, ANIMESH TRIVEDI, JONAS PFEFFERLE, AND CHRISTOS KOZYRAKIS

Ana Klimovic is a PhD student at Stanford University advised by Professor Christos Kozyrakis. Her research interests are in computer systems and computer architecture. She is particularly interested in improving performance and resource efficiency for cloud computing. Ana is a Microsoft Research PhD Fellow, Stanford Graduate Fellow, and Accel Innovation Scholar. anakli@stanford.edu

Yawen Wang is a second-year PhD student advised by Professor Christos Kozyrakis at Stanford University. She is broadly interested in computer systems and cloud computing. Her current research focuses on leveraging machine learning to manage cloud resources more efficiently. yawenw@stanford.edu

Patrick Stuedi is a researcher at IBM Research Zurich. His research interests are in distributed systems, networking, and operating systems. Patrick graduated with a PhD from ETH Zurich in 2008 and spent two years (2008–10) as a postdoc at Microsoft Research Silicon Valley. His work explores how modern networking and storage hardware can be exploited in distributed systems. Patrick is the creator of several open source projects such as DiSNI (RDMA for Java) and DaRPC (low latency RPC) and is co-founder of Apache Crail (Incubating). stu@zurich.ibm.com

Serverless computing platforms are increasingly being used to exploit massive parallelism and fine-grained billing for interactive analytics jobs [1–3]. A key challenge is exchanging intermediate data efficiently between tasks, as serverless tasks are short-lived and stateless. The systems commonly used to store and exchange intermediate data in serverless jobs today do not meet the performance, cost, and elasticity requirements of interactive analytics applications. We present *Pocket*, a fast, elastic, fully managed cloud storage service designed for efficient data sharing in serverless analytics applications. To achieve high performance and cost efficiency, Pocket leverages multiple storage technologies, right sizes resource allocations for jobs, and automatically scales cluster resources based on utilization. The system achieves similar performance to Redis, an in-memory datastore, while offering automatic, fine-grained scaling and significantly lower cost for serverless analytics jobs. Pocket is open source software [4].

Serverless Analytics

Serverless platforms like AWS Lambda, Google Cloud Functions, and Azure Functions enable users to quickly launch thousands of lightweight tasks, as opposed to entire virtual machines. Cloud providers automatically scale the number of serverless tasks based on application demands, and users pay only for the resources their tasks consume, at sub-second time granularity.

Though serverless computing was initially used for web microservices and IoT applications, its high elasticity and fine-grain billing make serverless computing appealing for more complex jobs, such as interactive analytics [1–3]. Analytics jobs typically consist of multiple stages of execution and require tasks in different stages to exchange data. We refer to the intermediate data shared between tasks as *ephemeral* (i.e., short-lived) data. We distinguish ephemeral data from the initial input and final output data of analytics jobs, which typically have longer lifetimes.

Traditional analytics frameworks (e.g., Spark) implement ephemeral data sharing with long-running framework agents buffering data in local storage on each node. In contrast, tasks in serverless deployments are short-lived, and any data that a task stores locally is lost when a task exits. Thus, direct communication between tasks is difficult, and the natural approach to share data is to use remote storage.

For instance, serverless analytics frameworks use object stores (e.g., S3), databases (e.g., CouchDB), or distributed caches (e.g., Redis).

Pocket: Elastic Ephemeral Storage for Serverless Analytics

Animesh Trivedi is a researcher at IBM Research Zurich. His interests are in anything and everything related to performance, ranging from multi-core CPUs to distributed environments. Currently, he is investigating how modern high-performance network and storage devices can be leveraged in large-scale data-processing systems such as Spark, Tensorflow, and serverless workloads. He is one of the founding members of the Apache Crail (Incubating) project. atr@zurich.ibm.com

Jonas Pfefferle is a Software Engineer at IBM Research Zurich in the Cloud Storage and Analytics group. His research interests are in virtualized distributed systems and datacenters, specifically in state-of-the-art network and storage technologies. Currently, he is working on exploiting high performance I/O devices with the focus on remote direct memory access (RDMA) and non-volatile memory (NVM) in data processing frameworks like Spark. Jonas holds a master's degree in computer science from ETH Zurich (2014). jpf@zurich.ibm.com

Christos Kozyrakis is a Professor in the Departments of Electrical Engineering and Computer Science at Stanford University. His research interests include resource-efficient cloud computing, energy-efficient computing and memory systems for emerging workloads, and scalable operating systems. Kozyrakis has a PhD in computer science from the University of California, Berkeley. He is a Fellow of the IEEE and ACM. kozyraki@stanford.edu

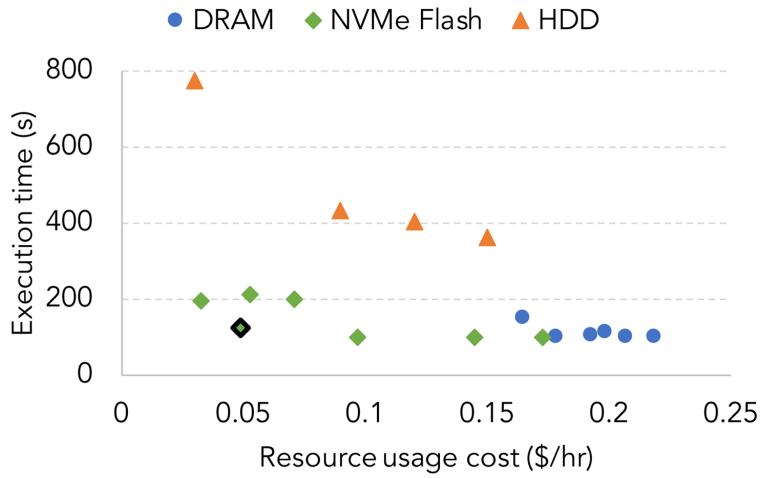

Figure 1: Example of the performance-cost tradeoff space for a serverless video analytics job using different storage technologies and VM types in Amazon EC2 for the ephemeral storage cluster. Data points of the same storage type represent applications using different numbers of nodes, compute resources, and network bandwidth.

However, existing storage services are not a good fit for sharing ephemeral data in serverless applications [5]. Popular fully managed cloud storage services, such as Amazon S3, are designed to store data with high durability and are not optimized for low latency or high elasticity. Distributed key-value stores offer good performance but burden users with managing the configuration and scale of a storage cluster. Selecting storage resource configurations for jobs is difficult yet critical for performance and cost efficiency [6]. Figure 1 shows an example of the performance-cost tradeoff for a serverless video analytics application using an ephemeral storage cluster configured with different storage technologies (DRAM, Flash, and disk), number of nodes, compute resources per node, and network bandwidth. Finding the minimum cost storage cluster configuration that provides optimal performance (e.g., the bold point in Figure 1) is nontrivial and gets even more difficult with multiple jobs.

Ephemeral Storage Requirements for Serverless Analytics

We study the ephemeral storage requirements of three different types of serverless analytics applications: distributed software compilation, video object recognition, and MapReduce sort. Figure 2 shows that ephemeral object size varies from 100s of bytes to 100s of megabytes. Hence, serverless analytics applications require both low latency, which is important for small object accesses, and high throughput, which is important for large object accesses. As serverless computing platforms elastically scale the number of tasks based on load, the ephemeral datastore must also be able to scale up and down automatically to meet dynamic I/O requirements while minimizing cost. In addition to rightsizing storage cluster resources based on current load, the system must place data on the right type of storage technology for each job by taking into account the latency, throughput, and cost tradeoffs of different technologies.

On the other hand, fault tolerance is not a high requirement for the ephemeral datastore as the data is short-lived, and application frameworks typically have built-in mechanisms, such as lineage tracking, that can be used to regenerate ephemeral data. Figure 3 shows the object lifetime CDF for the three serverless analytics jobs we study. Most ephemeral data objects only need to be stored for less than 30 seconds.

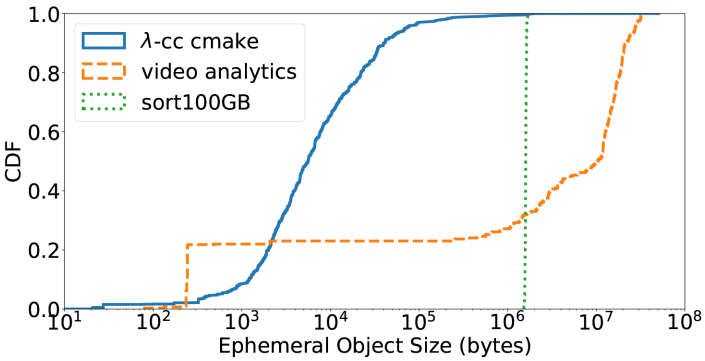

Figure 2: Ephemeral data object size CDF for three different serverless analytics applications. Objects vary widely in size.

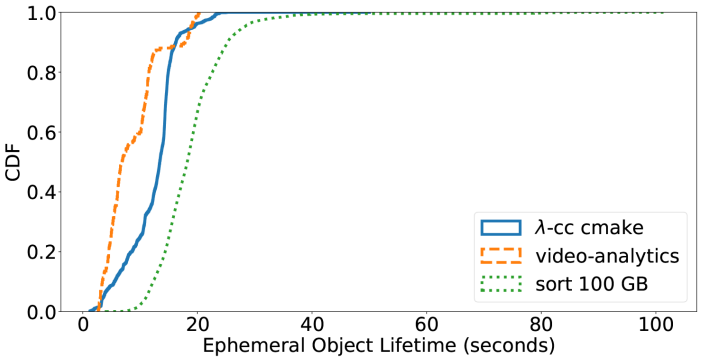

Figure 3: Ephemeral data objects have short lifetimes (seconds to minutes).

Ephemeral Storage as a Service

We introduce *Pocket*, an elastic storage service for ephemeral data sharing. The system provides high I/O performance while minimizing cost by leveraging multiple storage technologies with different performance-cost tradeoffs, rightsizing resource allocations for jobs, and automatically scaling cluster resources based on utilization. Pocket is a distributed `/tmp` for the cloud.

Pocket splits responsibilities across three separate planes: a control plane that determines data placement policies for jobs, a metadata plane that manages distributed data placement, and a metadata-oblivious data plane responsible for storing data. Pocket scales all three planes independently at fine granularity based on the current load. The system leverages optional hints about job characteristics, which can be provided by application frameworks or users via Pocket's API, to allocate the right storage technology, capacity, bandwidth, and CPU resources for each job. We intend for cloud operators to run Pocket as a fully managed storage service and charge users for only the storage capacity and bandwidth that their tasks consume.

Figure 4 shows Pocket's system architecture and how a job interacts with Pocket. To use Pocket, a job starts by registering with a logically centralized controller, which runs the control plane logic for Pocket. The controller decides the storage throughput, capacity, and type of storage technology to allocate for the job, leveraging any optional hints provided about the job's characteristics, such as latency sensitivity and the peak number of concurrent tasks. The controller decides on a data placement policy for the job, spinning up additional storage or metadata nodes if necessary to meet the job's allocation. The controller communicates the data-placement policy for the job to metadata servers, which will enforce data placement by routing client write requests. After registering with the controller, the job launches serverless tasks (i.e., lambdas), which issue GET/PUT requests using Pocket's client library.

Metadata servers route I/O requests to the appropriate storage nodes based on the job's data-placement policy determined upfront by the controller during job registration. By default, a job's ephemeral data is deleted when the job deregisters with the controller. However, Pocket's API accepts hints to manage data lifetime. For example, since we find that most ephemeral data is written and read only once, a user or application framework can hint to Pocket that an object should be deleted immediately after it is read, optimizing garbage collection.

In addition to rightsizing resource allocations across multiple dimensions upfront when jobs register, the controller also continuously monitors resource utilization in the cluster. Pocket's controller adds/removes nodes to keep CPU, storage capacity, and network bandwidth utilization within a target range. To balance load in a cluster of changing size, Pocket leverages the short-lived nature of ephemeral data and serverless jobs. We find that ephemeral data objects in the serverless applications we study typically only need to be stored for less than 30 seconds. Hence, migrating this data to redistribute load when nodes are added or removed would have high overhead. Instead, Pocket focuses on steering data for incoming jobs across active and new storage nodes in the cluster, while allowing nodes that the controller wants to take down to be drained as their data is garbage collected.

Implementation. Pocket's implementation leverages several open-source systems, and Pocket is also open source [4]. The metadata and data planes are built on top of the Apache Crail distributed datastore, which is designed for low latency, high throughput access to data with low durability requirements [7, 8]. Though Crail is originally designed to leverage RDMA networks, the system's modular architecture supports pluggable RPC libraries and storage tiers. We implement a TCP-based RPC library for Pocket. Our implementation of Pocket includes three different storage tiers. The first is a DRAM tier that efficiently serves client requests over TCP connections. The second tier is an NVMe Flash storage tier. We implement this tier using

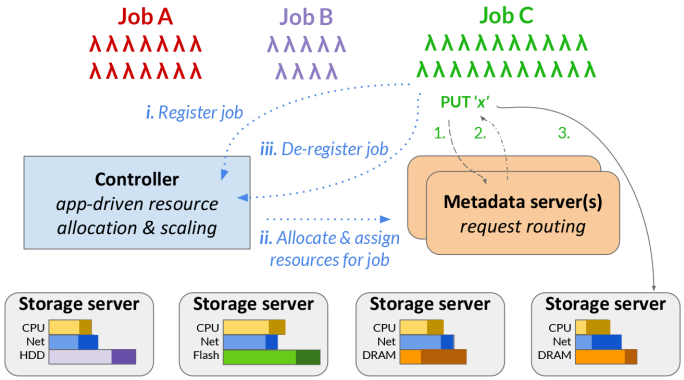

Figure 4: Pocket system architecture and the steps to register job C, issue a PUT from a lambda, and deregister the job. The shaded/colored bars on storage servers show used and allocated resources for all jobs in the cluster.

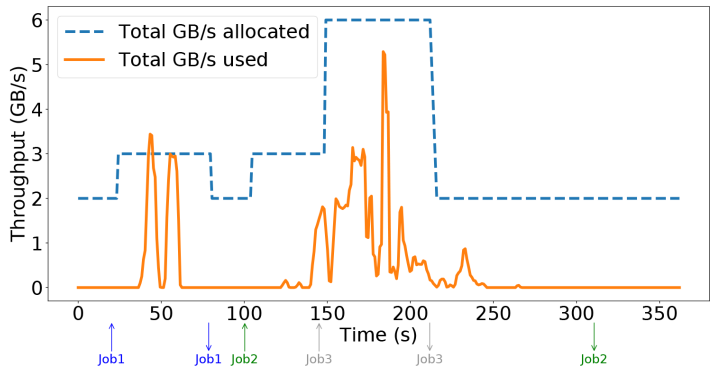

Figure 5: Pocket dynamically scales cluster resources to meet I/O requirements as jobs come and go.

ReFlex, a software system that enables fast, predictable access to remote Flash storage over commodity Ethernet networks [9]. The third tier is a generic block storage tier that allows Pocket to use any block device such as a hard-drive disk or SATA/SAS SSD with a standard kernel block device driver. We deploy Pocket storage and metadata servers inside of containers on AWS EC2 machines. We use Kubernetes to orchestrate the containers and implement autoscaling.

Elastic and Automatic Resource Scaling with Pocket

We evaluate Pocket with three different serverless analytics applications: a video analytics application that does object recognition, a MapReduce sort job, and a distributed compilation job that compiles the source code for `cmake`. The applications differ in their degree of parallelism, ephemeral object size distribution, and throughput requirements. We use AWS Lambda as our serverless computing platform.

Figure 5 shows how Pocket elastically scales cluster resources as multiple jobs register and deregister with the controller. In this experiment, we assume Pocket receives hints about the capacity and throughput requirements of each job. The first job is a 10 GB sort requesting 3 GB/s throughput. The second job does video object recognition, requesting 2.5 GB/s, and the third job is a different invocation of a 10 GB sort also requesting 3 GB/s. Pocket quickly and automatically scales the allocated storage bandwidth (dotted line) to meet application throughput demands (solid line). Application throughput briefly surpasses the total allocated throughput due to bursty EC2 VM network bandwidth, which causes a storage node to provide greater than the anticipated 1 GB/s bandwidth per node for a short period of time. In this experiment, the controller is configured to maintain a minimum cluster size of two storage nodes, which provides 2 GB/s cumulative throughput.

Comparing Pocket to Amazon S3 and Redis

We compare Pocket to two popular storage systems used by serverless analytics applications today. Amazon S3 is a fully managed cloud storage service that offers a convenient "serverless" storage abstraction and cost model in which users pay only for the capacity and bandwidth their tasks consume. S3 offers durable storage and is optimized for access to large objects. In contrast, Redis is a popular key-value store that uses DRAM to provide high performance. However, users need to manually select and manage resource configurations for a Redis storage cluster. Although Amazon and Azure offer managed Redis clusters through their ElastiCache and Redis Cache services, respectively, they do not automate storage management as desired by serverless applications. Users must still select instance types with the appropriate memory and compute and network resources to match their application requirements.

We first compare job execution time when using Pocket versus S3 and ElastiCache Redis as the ephemeral datastore. Figure 6 plots the per-task execution time breakdown for a 100 GB MapReduce sort job, run with 250, 500, and 1000 concurrent lambdas. The light-gray/orange bars show the time spent fetching original input data and writing final output data to long-term S3 storage, while the darker-gray/blue bars compare the time for ephemeral data I/O, comparing S3, Redis, and Pocket. S3 does not provide sufficient throughput for this I/O-intensive job, hence in the 250 lambda experiment, each task spends over three times longer shuffling data when using S3 compared to Redis or Pocket. When the job is run with 500 or more lambdas, S3 does not support sufficient request rates. The system returns errors and advises to reduce the I/O rate. On the other hand, Pocket provides similar throughput to Redis. In this experiment, we assume Pocket receives a hint that the job is not sensitive to latency—hence, Pocket uses NVMe Flash instead of DRAM. Thus Pocket achieves similar performance to Redis while dramatically saving cost.

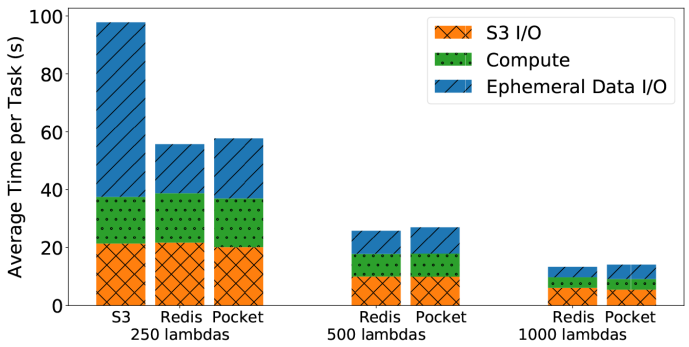

Figure 6: Average execution-time breakdown of each task (lambda) in a 100 GB sort job, run with 250, 500, and 1000 concurrent tasks

To compare the cost of running jobs using Pocket versus S3 and ElastiCache Redis for ephemeral data sharing, we derive a fine-grain resource cost model for Pocket based on Amazon EC2 pricing. Our minimum-size Pocket cluster, consisting of one controller node, one metadata server, and two NVMe Flash storage nodes, costs $1.63 per hour to run on Amazon EC2. However, Pocket's fixed cost can be amortized because the system is designed to support multiple concurrent jobs from one or more tenants. We intend for Pocket to be operated by a cloud provider and offered as a storage service with a pay-what-you-use cost model for users, similar to the cost model of serverless computing platforms. Hence, for our cost analysis, we derive fine-grain resource costs, such as the cost of a CPU core and the cost of each storage technology per GB, based on the prices of various EC2 instances.

Using this fine-grain resource pricing model for Pocket, we compare the cost of running the 100 GB sort, video analytics, and distributed compilation jobs with S3, ElastiCache Redis, and Pocket. For S3, we assume its GB-month cost is charged hourly. We base Redis costs on the price of entire VMs, not only the resources consumed, since ElastiCache Redis clusters are managed by individual users rather than cloud providers. Pocket achieves the same performance as Redis for all three jobs while saving 59% in cost. S3 is still orders of magnitude cheaper. However, S3's cloud provider-based cost is not a fair comparison to the cloud user-based cost model we use for Pocket and Redis. Furthermore, while the distributed compilation job has similar performance with all three ephemeral storage systems because it saturates CPU resources on serverless tasks, the execution time is 40 to 65% higher with S3 compared to Pocket for the video analytics and MapReduce sort jobs. A more detailed analysis of Pocket's performance and cost can be found in our OSDI '18 paper [10].

Conclusion

General-purpose analytics on serverless infrastructure presents unique opportunities and challenges for performance, elasticity, and resource efficiency. We analyzed the challenges associated with efficient data sharing and presented Pocket, a fully managed ephemeral data storage service. Pocket provides a highly elastic, cost-effective, and high performance storage solution for analytics workloads. Pocket achieves these goals using a strict separation of responsibilities for control, metadata, and data management. Although we designed Pocket specifically to enable efficient data sharing in serverless analytics applications, more generally, Pocket is a distributed temporary datastore that can be useful for a variety of different cloud applications.

Acknowledgments

We thank our OSDI shepherd, Hakim Weatherspoon, and the anonymous reviewers for their helpful feedback. We thank Qian Li, Francisco Romero, Sadjad Fouladi, and Nick Murphy for insightful technical discussions. This work is supported by the Stanford Platform Lab, Samsung, and Huawei. Ana Klimovic is supported by a Stanford Graduate Fellowship. Yawen Wang is supported by a Stanford Electrical Engineering Department Fellowship.

References

[1] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, pp. 363–376: https://www.usenix.org/system/files/conference/nsdi17/nsdi17-fouladi.pdf.

[2] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the Cloud: Distributed Computing for the 99%," in *Proceedings of the Symposium on Cloud Computing (SOCC '17)*, pp. 445–451.

[3] Databricks Serverless, "Next Generation Resource Management for Apache Spark": https://databricks.com/blog/2017/06/07/databricks-serverless-next-generation-resource-management-for-apache-spark.html, 2017.

[4] Pocket: https://github.com/stanford-mast/pocket, 2018.

[5] A. Klimovic, Y. Wang, C. Kozyrakis, P. Stuedi, J. Pfefferle, and A. Trivedi, "Understanding Ephemeral Storage for Serverless Analytics," in *Proceedings of the USENIX Annual Technical Conference (ATC '18)*, pp. 789–794: https://www.usenix.org/system/files/conference/atc18/atc18-klimovic-serverless.pdf.

[6] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics," in *Proceedings of the USENIX Annual Technical Conference (ATC '18)*, pp. 759–773: https://www.usenix.org/system/files/conference/atc18/atc18-klimovic-selecta.pdf.

[7] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas, "Crail: A High-Performance I/O Architecture for Distributed Data Processing," *IEEE Data Engineering Bulletin* 40, pp. 38–49.

[8] Apache Crail (Incubating): https://crail.incubator.apache.org, 2018.

[9] A. Klimovic, H. Litz, and C. Kozyrakis, "ReFlex: Remote Flash ≈ Local Flash," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17)*, pp. 345–359.

[10] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, C. Kozyrakis, "Pocket: Elastic Ephemeral Storage for Serverless Analytics," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*, pp. 427–444: https://www.usenix.org/system/files/osdi18-klimovic.pdf.