A Trace-driven Performance Evaluation of Hash-based Task Placement Algorithms for Cache-enabled Serverless Computing

Sacheendra Talluri
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
s.talluri@vu.nl

Nikolas Herbst
University of Würzburg
Würzburg, Germany
nikolas.herbst@uni-wuerzburg.de

Cristina Abad
ESPOL
Guayaquil, Ecuador
cabadr@espol.edu.ec

Animesh Trivedi
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
a.trivedi@vu.nl

Alexandru Iosup
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
a.iosup@vu.nl

ABSTRACT

Data-driven interactive computation is widely used for business analytics, search-based decision-making, and log mining. These applications' short duration and bursty nature makes them a natural fit for serverless computing. Data processing serverless applications are composed of many small tasks. Application tasks that use remote storage encounter bottlenecks in the form of high latency, performance variability, and throttling. Caching has been used to mitigate this bottleneck for intermediate data. However, the use of caching for input data, albeit widely used in industry, has yet to be studied. We present the first performance study of scaling, a key feature of serverless computing, on serverless clusters with input data caches. We compare 8 task placement algorithms and quantify their impact on task slowdown and resource usage before and after scaling. We quantify the consequences of using work stealing. We quantify the performance impact of scaling in the buffer period immediately after scaling. We find up to a 420% increase in task slowdown after scaling without work stealing and a 22% slowdown with work stealing. We also find that cache misses after scaling can lead to an additional 21% resource usage.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Theory of computation → Caching and paging algorithms**; • **General and reference → Performance**.

KEYWORDS

serverless, caching, scheduling, performance

ACM Reference Format:
Sacheendra Talluri, Nikolas Herbst, Cristina Abad, Animesh Trivedi, and Alexandru Iosup. 2023. A Trace-driven Performance Evaluation of Hash-based Task Placement Algorithms for Cache-enabled Serverless Computing. In *20th ACM International Conference on Computing Frontiers (CF '23), May 9–11, 2023, Bologna, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3587135.3592195

CF '23, May 9–11, 2023, Bologna, Italy

ACM ISBN 979-8-4007-0140-5/23/05.
https://doi.org/10.1145/3587135.3592195

1 INTRODUCTION

Business analytics, search-based decision-making, and other data-driven workloads with near-interactive deadlines require their computation to complete within short time spans. The short duration and frequently changing quantity of resources required [39] form a natural fit for serverless computing, which allows users to lease only the necessary resources for a short time [2, 13, 17]. Both academia (e.g., Lambada [26], Starling [30]) and industry (e.g., Snowflake [39], Databricks [4]) have recognized this synergy and have proposed systems to leverage it. However, storage has emerged as a bottleneck [9, 34] for such systems. Caches have been used to mitigate this bottleneck for *intermediate data* generated by applications during their execution and then used only once [21, 31]. In contrast, caching *input data* that gets reused multiple times, albeit much used in industrial serverless systems such as Snowflake [39], Databricks [1] and others [3, 5, 40] has yet to be studied. Input caching serverless systems often use hash-based placement algorithms such as consistent hashing [19] to decide data location and application execution location. In this work, we characterize the performance of input caching serverless clusters using hash-based placement algorithms using trace-driven simulation.

Interactive data processing workloads are composed of jobs. Jobs in these interactive workloads are short with a median runtime of 1 second and a 90th-percentile runtime of 7 seconds (from the dataset used in [39]. Supported by [22, 32]). The jobs are composed of tasks which run for 100s milliseconds. Tasks reading from remote storage such as AWS S3, common in data processing workloads, experience a storage bottleneck in the form of an additional 13 ms latency, with a 99th-percentile latency of over 1,200 ms [9]. The 99th-percentile latency is over ten times the duration of the average task and is longer than the duration of the median job. The read bandwidth is also highly variable [9, 31, 34] and reads are even throttled.

Interactive data processing applications often reuse data. The reuse can be due to the user exploring the data using different queries, or due to different users accessing the same data. Cloud storage access pattern analysis reveals that data reuse is common [14]. To take advantage of this data reuse, and remedy the storage bottleneck of remote cloud storage, serverless data processing platforms such as Snowflake [39], Databricks [1] and others [3, 5, 40], use a

local cache on each node. While remedying the storage bottleneck, the cache-enabled nodes raise a *scheduling* challenge. Objects accessed by tasks need node assignments, and tasks which access an object need the same node assignments repeatedly for caching to be effective. It is common for clusters which use cache-enabled nodes to use a hash-based placement policy such as ring-based consistent hashing [5, 39] and rendezvous hashing [3]. The placement policy hashes the object identifier and always redirects a task reading a certain data object to the same node.

The workload intensity in data processing clusters can change by an order of magnitude over a short period of time [39]. The number of nodes in the cluster are increased or decreased to match workload intensity using an autoscaler. When the number of nodes in the cluster changes, the data to node mapping changes, and tasks are directed to different nodes. Cluster scaling, changing the number of nodes, is a common operation in serverless clusters. While many hash-based placement algorithms were evaluated for object load balancing performance [7, 10, 11, 25, 28], none consider the impact on data processing task performance. We are the first to quantify the performance slowdown experienced by tasks scheduled onto a cluster using hash-based scheduling as cluster size changes.

A common problem when using hash-based task placement is an imbalance in task load allocated to nodes. Work stealing is a popular technique to help with this problem. Work stealing reallocates tasks from busy nodes to free nodes. The typical trade-off with work stealing is that it reduces response times but increases cache misses. We evaluate the impact of work stealing on task performance.

The key contributions of this work are:

(1) We design and implement a trace-based cache-enabled serverless system simulator in the OpenDC [23] datacenter simulator to evaluate the performance of hash-based task placement algorithms. The simulator allows us to assess the performance of tens of traces on hundreds of scenarios. We describe the system model we implement in Section 2, and the experimental setup we implement in Section 3.
(2) We evaluate the performance of cache-enabled serverless clusters when using 6 hash-based task placement algorithms in Section 4. We evaluate the performance before and after scaling events, as scaling is common in serverless clusters. We quantify the impact of combining work stealing with hash-based task placement algorithms. We quantify both the performance gain and the decrease in hit rate.
(3) We quantify the additional resource usage incurred by serverless clusters after scaling due to cache misses caused by data movement in Section 5. The quantification allows practitioners to design better autoscalers taking the additional resource consumption after scaling into account.
(4) We analyze the time immediately after a scaling event, and compare it to the slowdown during the rest of the time after the scaling event in Section 6. The comparison allows us to identify if there is any unique behavior that occurs immediately after scaling that increases task slowdown.

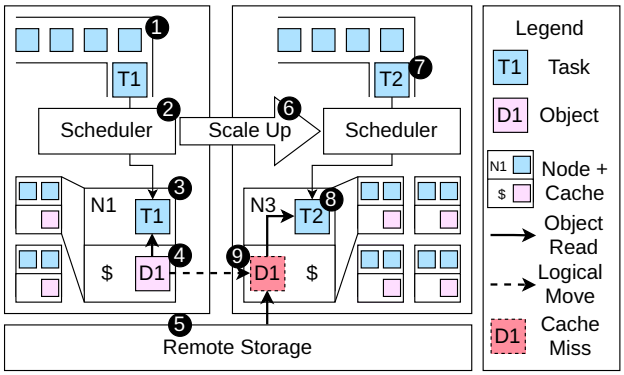


Figure 1: A server data processing cluster with per-node cache, scaling from 2 to 3 nodes. The logical location of some data objects is moved during scaling. Tasks which access the moved objects have to read it from remote storage leading to a cache ($) miss.

2 SERVERLESS DATA PROCESSING CLUSTER SYSTEM MODEL

In this section, we first describe the data processing cluster system model, and the technical terms we will use throughout the rest of the paper.

2.1 System Model

Figure 1 depicts the system model of a serverless data processing cluster. Data processing applications are structured as workflows of *tasks* (❶ in Figure 1). The scheduler ❷ schedules the tasks to run on a cluster of *nodes* ❸ (servers). The tasks read *data objects* ❹ they process from *remote storage* ❺ (e.g.: AWS S3, IBM COS, etc.). Remote storage has concurrency limits leading to throttling and high tail latency, which increase application runtime. Many serverless data processing platforms use a *local cache* $ on each node in the cluster to avoid the increased runtime. The scheduler uses a hash-based placement algorithm to direct all tasks which require a particular object to the same node. The object identifier of the object the task needs to access is hashed and used by the scheduler to direct tasks. A key feature of serverless clusters is that the number of nodes in the cluster *scales* ❻, increases or decreases, according to the workload intensity.

Scaling, up and down, causes the placement algorithm to change the object to node mappings so that the number of objects mapped to each node is balanced. When the mapping changes, the objects are not actually moved, it is only a *logical move* (Node 1 to Node 3 in Figure 1). Only the object to node mapping is updated. When subsequent tasks ❼ access the moved object, they are directed to the newly mapped node ❽ by the scheduler. This new node will not have the object in its local cache ❾. The object will be read from remote storage by the task.

A key characteristic of the system model that influences the results of our experiments is latency to remote storage. We assume a simple network topology where all nodes in the cluster are part of the same network with full bidirectional bandwidth between each other. The cluster is connected to a remote storage service with worse latency than the latency between the cluster

nodes [9, 34]. Specifically, for our experiments, we use the latency figures from the Pixels paper [9]. A more complex cluster topology with multiple storage and network tiers would result in more complete performance characterization. But, we limit the scope of our work to comparing hash-based algorithms. This means our results are only applicable to a two-level network topology with the network characteristics similar to ones we use. We believe the network characteristics we use representative of the remote storage access latency at this time.

However, network characteristics can change over time and invalidate our results. We release our simulation and analysis tools as open source software so that our experiments can be repeated with a simple change in network parameters as systems evolve and their latency characteristics change.

2.2 Terms and Metrics to Understand Serverless Data Processing Clusters

We now describe the terms and metrics we use to understand the performance of a data processing workload on a serverless cluster.

Scaling up is increasing the number of nodes in the cluster to deal with an increase in workload intensity. The increased workload intensity can be due to a higher number of tasks per second, increased average task duration, increased storage latency, increased average task resource requirements, or other reasons. *Scaling down* is decreasing the number of nodes in the cluster to deal with a decrease in workload intensity. The reasons for the decrease are the same as the reasons for the increase, except in the opposite direction.

All the nodes in a cluster do not process an equal portion of the workload. Making sure that each node processes an equal portion of the workload (load balancing) is an NP-hard problem, even with perfect information. The imperfect object distribution due to hash-based object placement and the load unawareness of the placement algorithms only adds to the difficulty.

Imbalance is the difference between the portion of the workload processed by a node compared to the average workload processed by the cluster. We quantify peak imbalance during a specific time period, such as a minute, five minutes, or the whole workload duration. We quantify imbalance during a period using the ratio of the maximum active workload allocated to any node in the cluster and the mean workload of all nodes in the cluster during that time period.

Tasks scheduled onto a node can experience a *cache miss* if the object they need to read is not in the local cache of the assigned node. We consider three categories of cache misses in this work. *Compulsory cache miss* is the absence of the object from the cache because it is the very first time it is accessed. *Capacity cache miss* is the absence of the object that was present earlier in the cache, but was evicted to make space for other objects. *Scaling cache miss* is the absence of the object from the cache because the object was remapped to a new node due to scaling and does not exist yet in the cache of the new node.

We quantify the performance of a task using the *slowdown* metric. We define slowdown as the ratio of the actual makespan of a task to the ideal makespan in the input trace. The actual makespan includes the wait time for the task before an empty slot becomes available for it to execute, the duration supplied in the trace (ideal makespan), and additional time to read an object from remote storage if it is a cache miss. A lower value of slowdown is better. It means that the task was completed close to its ideal duration.

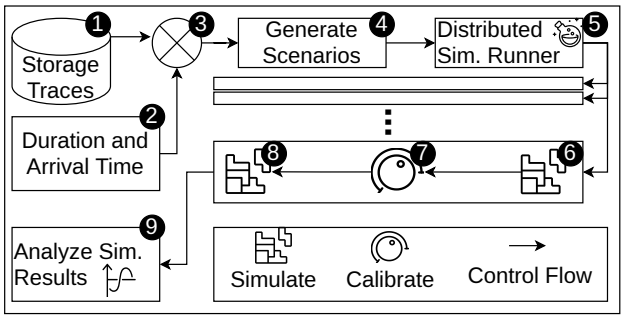


Figure 2: The experimental process we use to generate traces, and calibrate and run experiments.

Hash-based task placement algorithms, combined with a cache, reduce the number of reads from tasks to remote storage. But, they also constrain the nodes a task can be scheduled to. In Section 4 we quantify the slowdown of tasks when using hash-based task placement and compare it to slowdown using greedy and random data oblivious task placement algorithms.

3 EXPERIMENTAL METHOD TO DETERMINE THE IMPACT OF SCALING ON CACHE-ENABLED SERVERLESS CLUSTERS

We use trace-based simulation to study the impact of cluster scaling on task slowdown and workload resource usage. Our experiment setup consists of a trace generator which derives simulable traces from publicly available traces, the simulator that simulates the generated traces, and programs to analyze the output of the simulator. Our experimental process is depicted in Figure 2

For the input traces, we combine (❸ in Figure 2) the sequence of object identifiers from the real-world IBM COS traces [14] ❶ with a real-world inspired distribution [9] ❷ of task durations and interarrival times. Controlling the task durations and interarrival-times enables us to control resource utilization in the cluster under simulation. A controlled resource utilization helped us run controlled experiments on scaling behavior which would be difficult with a fully realistic trace which, has non-uniform resource usage.

The IBM COS traces [14] are the second largest publicly available set of storage traces at the time of writing. The IBM COS dataset consists of 99 different traces. Only 31 of them had a length greater than 2.8 million reads which we required for our simulations. The trace generator takes the 31-longest IBM COS traces and the scenarios we want to simulate as the input and combines them with task durations sampled from a realistic distribution [22, 32, 39]. We sample the task duration from a Pareto distribution with shape factor 1.44, with a median task duration of 100 milliseconds. We issue either 120 or 240 tasks per second, depending on the workload intensity we target. The simulated time duration of the trace is around 2.5 hours. We generate simulable traces for three scenarios ❹: doubling workload intensity, halving the workload intensity, and constant workload intensity. Of the 31, we use 29 traces for

our simulations. The two excluded traces lead to suspicious results like median slowdown in the thousands. We plan to analyze them further in the future.

We use a distributed simulation runner ❺ to run many scenarios concurrently across a cluster. The runner takes the list of scenarios along with input file locations as the input, runs the distributed cache simulator, and produces the output files at the specified locations.

We implement our cache-enabled serverless cluster simulator (❻ and ❽), which simulates the system model in Section 2, using the OpenDC [23] open-source datacenter simulator. The simulator takes the simulable trace and the experiment parameters as the input. The experiment parameters include the object placement algorithm, the autoscaler, the manual scaler, the workstealer, and their respective options. We use six hash-based data-aware task placement algorithms: Ring [19], Rendezvous [38], Maglev [11], Multiprobe [7], Dx [10], and Anchor [25]. The hash-based algorithms assign a task to a node in the cluster based on the hash of the object name the task will read. We also use two data-oblivious algorithms: Greedy and Random to have a point of comparison. The greedy algorithm assigns a task to the cluster's least loaded node. The manual scaler triggers a scaling event of a user-defined magnitude at a user-defined time. The manual scaler enabled us to run our controlled experiment. The workstealer allows for a free node to run a task from another node with the longest wait queue. Work stealing is successful at multi-core scheduling [24] and at least one serverless data processing platform [39] uses it.

The simulator uses a pre-warming mechanism to populate the caches before the actual simulation. For the pre-warming, we run half the simulable workload to populate the caches before starting the simulation that we use for our analysis. We use infinite-sized caches for the node-local caches in the simulator. The large cache size is representative of the extremely large (100 GBs or even TBs) caches available to data processing serverless clusters [14, 39]. We observe that our results are not sensitive to cache size. The performance difference between an infinite-sized cache and a cache of 1000 objects was node was only 5% at the median and 3.5% at the 99th percentile. Using an infinite-sized cache means no capacity cache misses occur.

The lack of capacity cache misses means all the performance impact we observed can be solely attributed to scaling cache misses. This aligns with the paper's goal to characterize the performance impact of scaling. Running the simulations with a limited cache and classifying cache misses in post processing would complicate the setup. It would require deeply instrument the simulator such that the cache eviction component is aware of the scaling component. That would strongly couple all components and limit extensibility. We could also classify cache misses in analysis, but that would require duplicating the eviction mechanism outside the simulator and re-running the whole eviction process on the trace during analysis. The duplication is error prone and double the the compute time required by the simulation and analysis process.

We maintain the same resource utilization before and after the autoscaling events (80%). We use two techniques for this. First, we control the ideal resource use of the input workload by adjusting the number of tasks (workload intensity) per second. We go from 240 tasks per second to 120 if we need to half the workload for an experiment. But, we still have unpredictable resource utilization due to the reads from remote storage due to compulsory and scaling cache misses. To calibrate ❼ the post-scaling resource utilization, we first simulate without calibration ❻. For example, 11 nodes before scaling and 22 after scaling. Then, we measure the number of misses and storage delays incurred after scaling and adjust the number of nodes to maintain the desired utilization with the new workload ❽. For example, increase the nodes from 22 to 24 to account for the additional resource usage and maintain the same resource utilization. We analyze ❾ the additional resource required by a workload after scaling in Section 5.

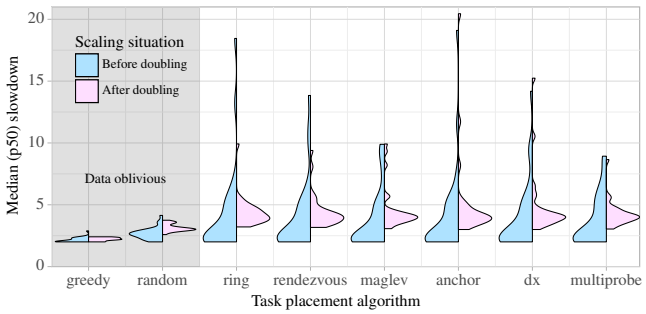
Apart from resource usage, the primary metric we measure during our analysis ❾ of simulation results is the slowdown. We analyze the p50 and p99 slowdown for different traces across all scenarios. We compare the slowdown before and after a cluster scaling for both an increase and a decrease in cluster size. We analyze the slowdown results in Section 4.

4 SLOWDOWN DUE TO HASH-BASED TASK PLACEMENT

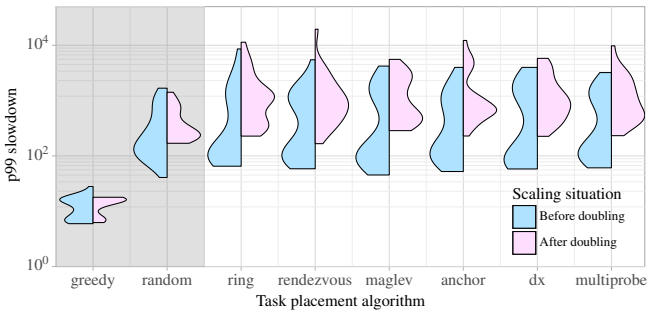
We analyze the performance of hash-based task placement algorithms using the slowdown metric with the following main observations:

- **O-1:** Hash-based task placement algorithms than the data oblivious load-based greedy algorithm, when we do not use workstealing.
- **O-2:** The median slowdowns caused by different hash-based task placement algorithms do not differ. The maximum varies only by about 2× for p99 tasks. The little variation might be the reason why simple ring based consistent hashing is widely used in commercial systems despite the existence of more modern algorithms.
- **O-3:** The median p50 and p99 task slowdown increases after scaling when using hash-based algorithms. The distribution of the p99 slowdown is wider, and the extremes are higher after scaling.
- **O-4:** Hash-based placement algorithms perform significantly better than the data-oblivious greedy algorithm when both are combined with work stealing.
- **O-5:** Work stealing decreases the median slowdown of tasks over 100× at the cost of 49% decrease in cache hit rate for p99 tasks.

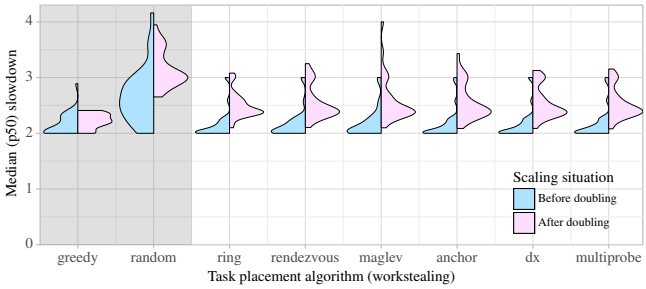
We derive our observations from Figure 3, which depicts p50 and p99 task slowdowns. The slowdowns are depicted before and after an increase in cluster size due to an increase in workload intensity. The new cluster size can vary based on resource usage, which we analyze in Section 5. In most cases, the cluster size at least doubles. The horizontal axis represents the task placement algorithm. The vertical axis represents the median or 99th percentile slowdown. For each placement algorithm, the distribution on the left depicts the slowdowns of the 29 traces before the cluster scales up. The distribution on the right depicts the distribution after the cluster scales up. Figures 3a and 3b depict the slowdown when not using work stealing along with the task placement algorithm. Figures 3c and 3d depict the slowdown when using workstealing.



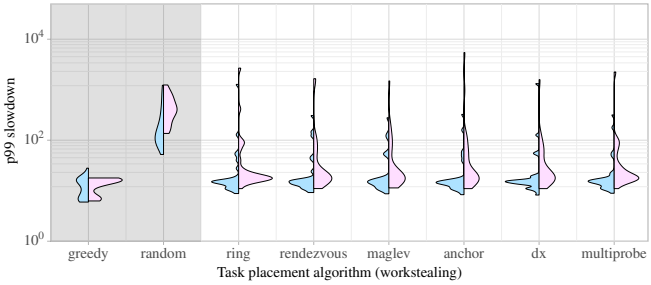
(a) Distribution of median slowdown across all traces.



(b) Distribution of 99th percentile slowdown across all traces.



(c) Distribution of median slowdown with work stealing.



(d) Distribution of 99th percentile slowdown with work stealing.

Figure 3: p50 and p99 slowdown across 29 traces, with and without work stealing. Lower slowdown is better.

We compare the slowdown experienced by tasks when using hash-based placement algorithms to data oblivious placement algorithms. We observe, in Figure 3a, only a 15% increase in p50 slowdown for hash-based algorithms compared to the data-oblivious load-aware greedy policy. But, we notice an overwhelming 43× increase in the median of p99 slowdowns (**O-1**). The median comparison even excludes the massive outliers which the greedy policy does not have. We find that the performance of hash-based algorithms is in-line with the performance of the random task placement algorithm.

Among hash-based placement algorithms, we do not find any significant performance difference as measured by the slowdown metric (**O-2**). This lack of significant difference might explain the popularity of simple ring based consistent hashing in production software systems [12, 39]. Newer algorithms such as Maglev [11] provide more efficient hash computations and slightly better load balancing properties. We notice this in the slightly better p50 slowdown distribution for Maglev compared to the simple ring hash. We conjecture that users find that the minor increase in performance is not worth the increased complexity. This might explain the low popularity of an algorithm like Maglev (**O-2**).

We observe in Figure 3a that the median of p50 slowdowns increases from 2.4 to 4 (40%) (**O-3**). This increase after scaling does not occur with data oblivious task placement algorithms. We investigate further and find that the large increase is caused by an increase in cluster imbalance due to hash-based placement and the consequent long wait times. Operators of autoscaling clusters with hash-based task placement algorithms should be aware that scaling up their cluster can lead to a significant increase in slowdown and task response times in some cases.

We observe in Figure 3b that the median of p99 slowdowns increases significantly from 173.9 to 912.2 (420%) (**O-3**). This is much higher than the p50 case where the change was only 40%. We also observe a similar increase in the case of halving cluster size.

Existing work only demonstrates that a decrease in cluster size causes a slight imbalance in object allocation [7, 10, 11, 25, 28, 35]. But, existing work does not study the impact of this object allocation imbalance on on the slowdown of tasks using the objects. We demonstrate that for most traces we studied, the p50 slowdown increases slightly and the p99 slowdown increases significantly with changing cluster size. But, the max slowdown increases greatly for some traces.

Key takeaway: The task slowdown, due to load imbalance, can be more than 10× the object imbalance for some traces. For these traces, both the p50 and p99 slowdown increase greatly (420%) after scaling. This increase is much more than what can be linearly extrapolated from an object imbalance of 6.6% to 42% for O(100k) objects in recent work [25]. We posit that the hash-based placement policies not being load-aware exacerbates the imbalance. An imbalance of a popular object increases the task imbalance disproportionately compared to the object imbalance.

Attributing Slowdown to Imbalance or Storage Delay

The slowdown metric is the ratio of actual task execution time to ideal execution time. Two sources of delay lead to a longer task execution time. One is the time a task waits to be scheduled, and the other is the time a task spends waiting for the remote storage to respond. The time a task spends waiting in a not fully saturated system, like ours, is due to the load imbalance between nodes in the system. We compare the wait time to the storage delay to attribute task slowdown to either imbalance or remote storage.

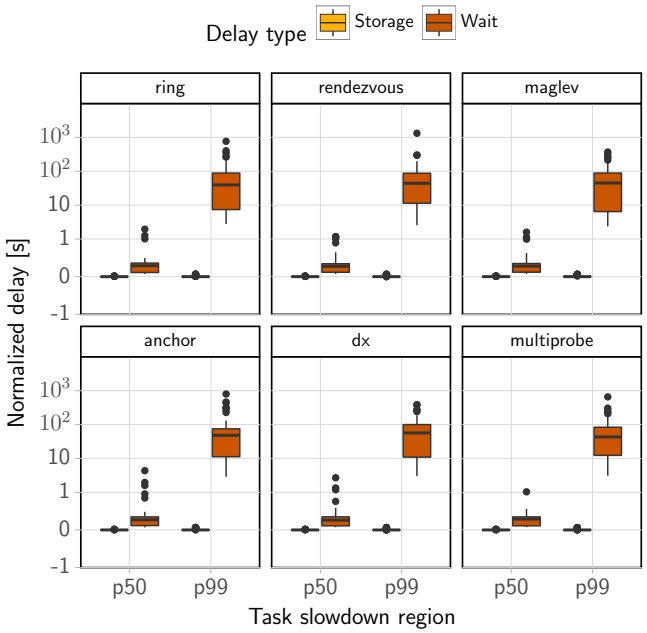


Figure 4: Normalized delay distributions for the p50 and p99 regions of the slowdown distribution.

Figure 4 depicts the normalized delay experienced by tasks in different regions of the slowdown distribution. The p50 region comprises of tasks that experienced slowdowns between the 45th percentile and the 55th percentile. The p99 region comprises tasks that experienced more slowdown than the 99th percentile task. We add the delays of all tasks in the region and divide them by the number of tasks to arrive at the cumulative delay. We then normalize the cumulative delay by dividing it by the number of tasks in the region. The normalization is necessary as the sizes of the two regions are different and each region has a different number of tasks. We refer to normalized wait time as the wait time and normalized storage delay as the storage delay for brevity.

We observe that the wait time is much higher than the storage delay. The median wait time is 10x higher than the median storage delay for p50 tasks. The median wait time is 100x higher than the storage delay for p99 tasks. The results indicate that imbalance is the major cause of slowdown. The low storage delay indicates that miss rate is not a major cause of slowdown. Therefore, it is beneficial to trade off some of that hit rate to reduce the imbalance. Work stealing is a technique that enables us to make this trade-off.

Impact of Workstealing

Workstealing is a popular technique to reduce resource waste in the cluster. The key idea behind workstealing is that free nodes steal tasks from heavily utilized nodes, thereby reducing the imbalance. Workstealing also reduces p99 slowdown caused by imbalance as tasks that are waiting in the queues of busy nodes are picked up by free nodes. Workstealing has been use serverless data analytics clusters [39], and has been demonstrated to be the best way to increase performance in multi-core clusters [24]. In this section, we analyze the performance of hash-based task placement algorithms when combined with workstealing.

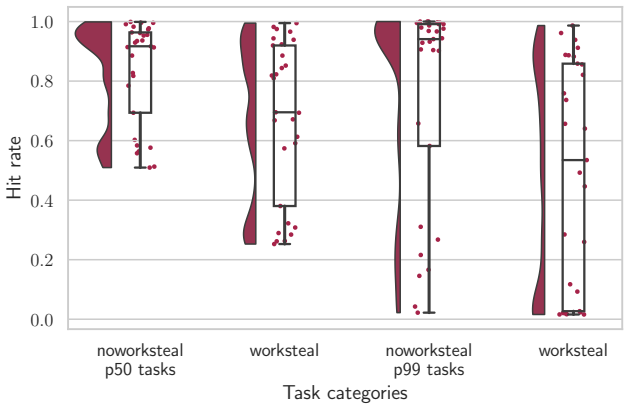


Figure 5: The impact of work stealing on hit rate for p50 and p99 tasks. Work stealing trades-off hit rate for a reduction in imbalance.

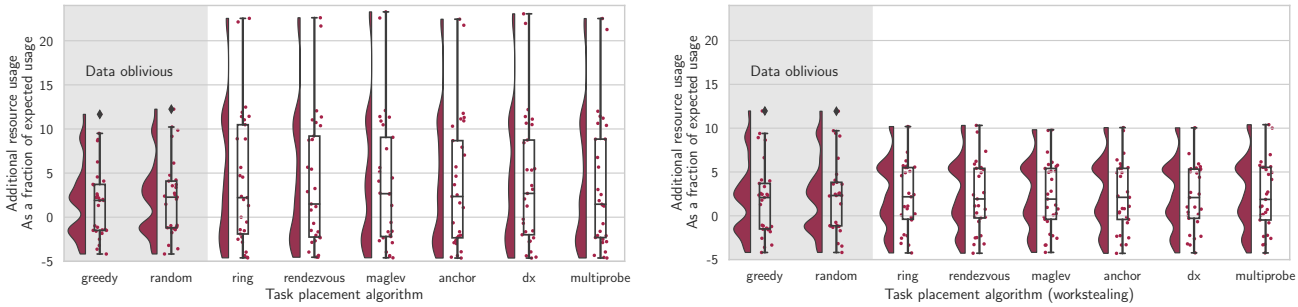
We observe in Figure 3c that the median of p50 slowdowns increases after increasing the cluster size from 2.1 to 2.4. The 14% increase lower than the 40% increase in the case without work stealing (**O-3**). Further analyzing the data, we found that the increased slowdown is due to increased cache misses after scaling. This is consistent with the results in existing literature for hash-based task placement algorithms. Once the slowdown due to imbalance is greatly reduced by work stealing, the only slowdown that remains is due to cache misses.

We observe in Figure 3d that the median of p99 slowdowns increases from 14.7 to 17.9 (22%)after scaling(**O-3**). This is much lower than the 420% in the scenario without work stealing. We observe similar behavior for the case of halving cluster size, which is not depicted here. Our observation is consistent with existing literature that moving objects due to a change in cluster size increases slowdown. The increased slowdown is due to the increased cache missed after scaling, similar to the p50 case. We also observe several outliers, visible as almost vertical lines in the graph, that occur for all hash-based placement algorithms. We do not yet have a reason as to why these outliers occur.

Work stealing improves cluster load balance by scheduling tasks from busy nodes onto free nodes. The rescheduling leads to a decrease in cache hit rate. We quantify the improvement in slowdown, and the subsequent decrease in cache hit rate. The reduction in hit rate for both p50 and p99 tasks in Figure 5. p50 tasks are tasks with a slowdown between the p45 slowdown and the p55 slowdown. p99 tasks are those slowdown above the p99 slowdown.

Work stealing decreases the median p50 slowdown from 3.9 to 2.4, a 60% improvement. Work stealing decreases the median hit rate for p50 tasks from 0.92 to 0.70, a 24% penalty. Work stealing decreases the median p99 slowdown over 100×. Work stealing decreases the hit rate of p99 tasks from 0.94 to 0.55, a 41% penalty (**O-5**). This much decreased hit rate could be a significant cause of the increased slowdown of p99 tasks while work stealing. Without work stealing, imbalance is the solely responsible for p99 task slowdowns.

Key takeaway: Hash-based task placement algorithms when combined with work stealing perform better than data oblivious greedy scheduling (**O-4**). This is a contrast to the clear success of



(a) Additional resource use after doubling the cluster size.

(b) Additional resource use after doubling, with work stealing.

Figure 6: Additional resources consumed apart from the expected doubling in cluster size when doubling the workload. The additional resource are required to maintain the cluster utilization before and after the workload doubled. Lower additional resource consumption is better.

data-oblivious greedy algorithm in the no work stealing scenario. Work stealing significantly decreases p99 slowdowns (100×), but with a significant cache hit rate penalty (41%).

5 INCREASED RESOURCE USE AFTER SCALING

A scaling event occurs when the workload being served by the cluster changes. If the workload doubles in intensity, then the number of resources in the cluster are be doubled to handle the workload. If the workload halves, then the number of resources are also halved. For the cache-enabled clusters we study, there is an additional resource cost due to the scaling cache misses that takes place when cluster size changes. The additional resource consumption is a combination of the miss rate and the latency distribution of the remote storage the objects are fetched from. We analyze this additional resource consumption with the following main observations:

- **O-6:** The resource consumption increases by up to 21% for many traces after scaling. For a few traces, the resource consumption decreases by up to 5%. The median increase is 2.5%.
- **O-7:** Work stealing brings down the maximum additional resources required from 21% to 10%.

We compute the additional resources, CPU-time in our case, consumed by the workload by first computing the expected resources consumed by the workload after it changes in magnitude. We then subtract the expected consumption from the actual consumption. For example, consider a workload which runs on 11 servers at 80% utilization. When the workload intensity doubles, we expect it to use 22 servers at 80% utilization. But, if it runs at 85% utilization, we add 2 more server make the configuration 24 servers at 80% utilization. The 2 additional servers required to equalize the utilization before and after scaling are additional resources.

We depict in Figure 6 the additional resources consumed by different traces, after doubling the workload intensity, as a function of the task placement algorithm used. Figure 6a depicts the additional resources used when not using work stealing. Figure 6b depicts the additional resources used when using work stealing. The violins and boxplots of the additional resource consumed per trace are depicted in the figures. Each point in the figure corresponds to a separate trace.

We observe in Figure 6a that a median of 2.5% additional resources are consumed at the median when using hash-based task placement policies. This is slightly lower than the 3% of the greedy policy. But, at the 75th percentile, 9% additional resource is consumed by hash-based policies. The additional resource consumption reaches 21% at the maximum end (**O-6**). We also observe that some traces experience lower resource consumption after scaling. The lower resource consumption is represented by all the values before 0%. We observe similar results for the case of halving the workload intensity, and consequently the worker size.

We observe in Figure 6b that a median of 2.4% additional resources are consumed when using hash-based task placement policies with work stealing. The additional resource use by the greedy policy remains the same 3% when using or not using work stealing. The 75th percentile additional resource consumption drops from 8% to 5.5% due to work stealing. The maximum resource consumption also drops from 21% to 10% due to work stealing (**O-7**). We observe similar results for the case of halving the workload intensity, and consequently the worker size.

Key takeaway: Cache-enabled serverless cluster can consume more resource after scaling (up to 21% more) due to cache misses caused by data movement. Autoscalers should take this additional resource use into account when deciding how much to scale.

6 IMMEDIATE IMPACT OF SCALING

The number of nodes in a cluster changes during a scaling event. Due to this membership change, tasks can now be mapped to different nodes than they were mapped to before the scaling event. The mapping change causes additional cache misses and increased slowdown due as newly mapped tasks read data from remote storage. The cluster state stabilizes after all the data is moved to the newly mapped locations. In this section, we investigate if performance characteristics immediately after a scaling event, during the buffer period, are different compared to stable period.

We define the 1 minute period immediately after a scaling event as *buffer period*. We define the rest of the period from the end of the buffer period to the last time a task is submitted as the *after period*.

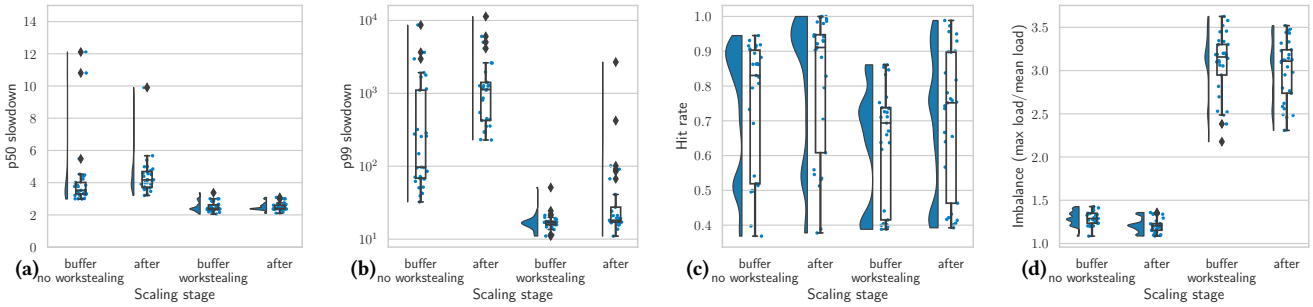


Figure 7: Comparing the 1 minute *buffer period* immediately after scaling, with the steady state period *after* the buffer period lasting till the end of the trace. Exemplary results for the ring task placement policy.

We depict the p50 slowdown, p99 slowdown, hit rate, and imbalance between the buffer period and the period after buffer period, for both configurations with and without work stealing in Figure 7. We only present results when using the ring task placement policy when increasing the cluster size. The performance results for other hash-based task placement policies and decreasing the cluster size follow a similar trend.

Figure 7a depicts the p50 slowdown for different configurations. Some outliers are present for the buffer period when not using work stealing are omitted as they are higher on the vertical axis maximum value of 15. We observe a median p50 slowdown of 3.5 for the buffer period compared to 4.2 for the after period, when not using work stealing. The key feature however are the extreme outliers in the buffer period. These findings imply that most traces do not see a significant performance drop during the buffer period, but some trace do experience a drop. When work stealing is enabled, we do not observe any performance difference between the buffer period and the after period.

We observe in Figure 7b that the p99 slowdown is higher during the after period than the buffer period, for both cases with and without work stealing. This is a contrast to the p50 slowdown case where the buffer period had higher slowdowns.

We investigate the reasons for the high p50 slowdown during the buffer period. We analyze the hit rate and the load imbalance across nodes during that period. But, we do not find a convincing explanation for the high p50 slowdown for some traces. We observe in Figure 7c that the buffer period has a lower hit rate than the after period, for both cases with and without work stealing. We also observe that the hit rate for the work stealing configurations is lower than their non-workstealing counterparts. The imbalance during the buffer period is higher than that during the after period, when not work stealing, as depicted in Figure 7d. Surprisingly, the imbalance for the configurations using work stealing is worse than configuration not using work stealing.

Key takeaway: Slowdown during the buffer period immediately after scaling is not much worse than the slowdown after the buffer period. We do however find some outlier traces which experience extreme slowdown during the buffer period.

7 FUTURE WORK

Validating the simulator: We use real-world object traces and real-world parameters for our task durations and remote storage network latency. But, the results we obtain are only indicative of real-world performance. Ideally, we would validate the simulator by running the experiments in a real world setup and comparing the results with the simulation results. Validation would allow readers to use our quantitative results in their results instead of just as indicators of performance trends. However, this paper required 990 simulations and validating them all would require 118,800 hours of CPU time. We are looking into validating the simulator on a subset of the experiments.

Other data-aware architectures: We specifically evaluate data-aware hash-based algorithms for task placement. But, other architectures based on metadata servers [27, 36], and hybrid hashing and metadata servers [29] exist. They also need to be comparatively evaluated for a full picture of data-aware scheduling for serverless computing.

Dynamic workload traces: We use a simple workload model with only one change in workload intensity per run. Experiments with dynamic traces which have many workload intensity variations per run can reveal more bout the behavior of placement policies in a real-world setting.

Evaluate multiple trace characteristics: In this work, we only evaluate the impact of scheduling policies on different traces. But, we only consider one characteristic in the trace, the object identifier. We need to further evaluate the impact of other trace properties such as the object size, request size, and inter-arrival time.

8 RELATED WORK

Hash-based object placement: The imbalance in the number of objects allocated to different servers in a cluster has been empirically studied for all hash-based object placement policies we looked at. It was either studied in the original work which proposed the policy [7, 10, 11, 25, 28], or in subsequent work which uses the policy [37]. A recent work [35] studies the load imbalance in the number of objects per server after a cluster size change, common in autoscaling serverless applications.

CH-RLU [15] and PASch [8] use consistent hashing to schedule schedule stateless function invocations. Both studies identify that using naive consistent hashing causes long queues due to load-imbalance, and propose a load-aware consistent hash to mitigate the problem.

We are the first to systematically quantify the slowdown of, and additional resources used by, data-intensive serverless applications, before and after autoscaling, caused by hash-based object placement

policies, and subsequent task scheduling based on those object locations.

Serverless/Autoscaling storage workloads: Industrial data processing software such as Snowflake [39], Databricks [4], Qubole [5], Quickwit [3], and Milvus [40] use input data caching. Snowflake uses consistent hashing with work stealing. Databricks uses greedy scheduling with bounded delay. Quickwit uses rendezvous hashing. Qubole uses consistent hashing.

Stateful serverless research prototypes Boki [16] and Jiffy [20] both use hashing to allocate keys to nodes in their key-value store implementations.

Caching has been used to mitigate the storage bottleneck for intermediate generated by serverless applications [21, 31]. Pixels [9] uses intelligent data placement to overcome the latency penalty of reading data from remote storage. Several works [26, 30, 34] propose mitigation strategies, complementary to caching, to overcome throttling and latency penalty of reading input data.

Cache-enabled serverless clusters make design decisions other than hash-based scheduling. We do not evaluate these designs in this work. An evaluation of such decisions would be valuable future work. Cloudburst [36] and OFC [27] use a central metadata server to perform cache-aware scheduling. We do not compare hash-based schedulers with schedulers using central metadata servers in this work. Faa$T [33] prefetches the data required by individual executions. We do not evaluate prefetching in this work.

Systematic exploration of scheduling policies: Hermod [18] systematically explores the policy space of load balancing and consolidation strategies for stateless serverless functions. There has been recent work on systematic evaluation of load balancing policies for tasks in a single-server multi-core setup [24]. It demonstrates the success of work-stealing as a load balancing policy in a multi-core setup. There is also a reference architecture for datacenter scheduling [6]. All three aforementioned works do not consider object location or any storage during task placement. We are the first to systematically the study the object of hash-based object placement on task placement, load balance, and slowdown.

9 CONCLUSION

Data-driven interactive computation is an important workload widely used for business analytics, search-based decision making, and log mining. These applications' short duration and bursty nature are a natural fit for serverless computing. These applications are composed of many tasks that run for 100s of milliseconds. Application tasks which read data from remote storage such as AWS S3 experience a storage bottleneck in the form of an additional 13 ms latency, with a 99th-percentile latency of over 1,200 ms [9]. The 99th-percentile latency is over ten times the duration of the average task.

Serverless data processing platforms such as Snowflake [39], Databricks [1] and others [3, 5, 40], remedy this bottleneck by using a local cache on each node. While remedying the storage bottleneck, the cache-enabled nodes raise a scheduling challenge. Objects, which tasks access, are assigned to nodes, and tasks which access an object need to be mapped to the same node repeatedly for caching to be effective. Hash-based task placement policies are commonly used by cluster to direct tasks to nodes with the appropriate data. The mapping from tasks to nodes changes as the scales, a common occurrence in serverless clusters. This change has a performance impact. We are the first to quantify the impact of scaling on task performance in cache-enabled data processing clusters.

We analyze task slowdown, hit rate, and imbalance before and after scaling. We analyze the additional resources used after scaling, above the expected change to serve the workload. We perform this evaluation using trace based simulation by running hundreds of scenarios using IBM COS [14] traces using a cache-enabled serverless cluster simulator implement in OpenDC [23].

The data, simulator, and auxiliary scripts are open-source and available on Zenodo at https://zenodo.org/record/7812238.

From our analysis, we extract 7 main observations, and the following 4 key findings:

(1) The task slowdown due to load imbalance can be more than 10× the object imbalance for some trace.
(2) Hash-based task placement algorithms only outperform data-oblivious greedy scheduling when combined with workstealing.
(3) Cache-enabled serverless cluster can consume more resources after scaling (up to 21% more) due to cache misses.
(4) Slowdown during the buffer period immediately after scaling is not much worse than the slowdown after the buffer period.

ACKNOWLEDGMENTS

This project is co-funded by the projects NWO Top2 OffSense, EU H2020 GraphMassivizer, and EU MCSA-RISE CLOUDSTARS.

REFERENCES

[1] 2022. Optimize performance with caching on Databricks. https://docs.databricks.com/optimizations/disk-cache.html. Accessed: 10-10-2022.
[2] 2022. Serverless: What it is. https://glossary.cncf.io/serverless/. Accessed: 10-10-2022.
[3] 2023. Architecture of Quickwit Full-text Search. https://quickwit.io/docs/concepts/architecture/. Accessed: 01-02-2023.
[4] 2023. Databricks Serverless Compute. https://docs.databricks.com/serverless-compute/index.html. Accessed: 01-02-2023.
[5] 2023. Quble Rubix. https://github.com/qubole/rubix. Accessed: 01-02-2023.
[6] Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, and Alexandru Iosup. 2018. A reference architecture for datacenter scheduling: design, validation, and experiments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*. IEEE / ACM, 37:1–37:15. http://dl.acm.org/citation.cfm?id=3291706
[7] Ben Appleton and Michael O'Reilly. 2015. Multi-probe consistent hashing. *CoRR* abs/1505.00062 (2015). arXiv:1505.00062 http://arxiv.org/abs/1505.00062
[8] Gabriel Aumala, Edwin F. Boza, Luis Ortiz-Avilés, Gustavo Totoy, and Cristina L. Abad. 2019. Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2019, Larnaca, Cyprus, May 14-17, 2019*. IEEE, 282–291. https://doi.org/10.1109/CCGRID.2019.00042
[9] Haoqiong Bian and Anastasia Ailamaki. 2022. Pixels: An Efficient Column Store for Cloud Data Lakes. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 3078–3090. https://doi.org/10.1109/ICDE53745.2022.00276
[10] Chaos Dong and Fang Wang. 2021. DxHash: A Scalable Consistent Hash Based on the Pseudo-Random Sequence. *CoRR* abs/2107.07930 (2021). arXiv:2107.07930 https://arxiv.org/abs/2107.07930
[11] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, Katerina J. Argyraki and Rebecca Isaacs (Eds.). USENIX Association, 523–535. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eisenbud

[12] Mostafa Elhemali, Niall Gallagher, Nick Gordon, Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somasundaram Perianayagam, Tim Rath, Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, and Akshat Vig. 2022. Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service. In *2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, Jiri Schindler and Noa Zilberman (Eds.). USENIX Association, 1037–1048. https://www.usenix.org/conference/atc22/presentation/elhemali

[13] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uta, and Alexandru Iosup. 2018. Serverless is More: From PaaS to Present Cloud Computing. *IEEE Internet Comput.* 22, 5 (2018), 8–17. https://doi.org/10.1109/MIC.2018.053681358

[14] Ohad Eytan, Danny Harnik, Effi Ofer, Roy Friedman, and Ronen I. Kat. 2020. It's Time to Revisit LRU vs. FIFO. In *12th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2020, July 13-14, 2020*, Anirudh Badam and Vijay Chidambaram (Eds.). USENIX Association. https://www.usenix.org/conference/hotstorage20/presentation/eytan

[15] Alexander Fuerst and Prateek Sharma. 2022. Locality-aware Load-Balancing For Serverless Clusters. In *HPDC '22: The 31st International Symposium on High-Performance Parallel and Distributed Computing, Minneapolis, MN, USA, 27 June 2022 - 1 July 2022*, Jon B. Weissman, Abhishek Chandra, Ada Gavrilovska, and Devesh Tiwari (Eds.). ACM, 227–239. https://doi.org/10.1145/3502181.3531459

[16] Zhipeng Jia and Emmett Witchel. 2021. Boki: Stateful Serverless Computing with Shared Logs. In *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, Robbert van Renesse and Nickolai Zeldovich (Eds.). ACM, 691–707. https://doi.org/10.1145/3477132.3483541

[17] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, João Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *CoRR* abs/1902.03383 (2019). arXiv:1902.03383 http://arxiv.org/abs/1902.03383

[18] Kostis Kaffes, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2022. Hermod: principled and practical scheduling for serverless functions. In *Proceedings of the 13th Symposium on Cloud Computing, SoCC 2022, San Francisco, California, November 7-11, 2022*, Ada Gavrilovska, Deniz Altinbüken, and Carsten Binnig (Eds.). ACM, 289–305. https://doi.org/10.1145/3542929.3563468

[19] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. 1997. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, Frank Thomson Leighton and Peter W. Shor (Eds.). ACM, 654–663. https://doi.org/10.1145/258533.258660

[20] Anurag Khandelwal, Yupeng Tang, Rachit Agarwal, Aditya Akella, and Ion Stoica. 2022. Jiffy: elastic far-memory for stateful serverless analytics. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis (Eds.). ACM, 697–713. https://doi.org/10.1145/3492321.3527539

[21] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 427–444. https://www.usenix.org/conference/osdi18/presentation/klimovic

[22] Zhenxiao Luo, Lu Niu, Venki Korukanti, Yutian Sun, Masha Basmanova, Yi He, Beinan Wang, Devesh Agrawal, Hao Luo, Chunxu Tang, Ashish Singh, Yao Li, Peng Du, Girish Baliga, and Maosong Fu. 2022. From Batch Processing to Real Time Analytics: Running Presto® at Scale. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 1598–1609. https://doi.org/10.1109/ICDE53745.2022.00165

[23] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin Van Eyk, Laurens Versluis, Vincent van Beek, and Alexandru Iosup. 2021. OpenDC 2.0: Convenient Modeling and Simulation of Emerging Technologies in Cloud Datacenters. In *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021, Melbourne, Australia, May 10-13, 2021*, Laurent Lefèvre, Stacy Patterson, Young Choon Lee, Haiying Shen, Shashikant Ilager, Mohammad Goudarzi, Adel Nadjaran Toosi, and Rajkumar Buyya (Eds.). IEEE, 455–464. https://doi.org/10.1109/CCGrid51090.2021.00055

[24] Sarah McClure, Amy Ousterhout, Scott Shenker, and Sylvia Ratnasamy. 2022. Efficient Scheduling Policies for Microsecond-Scale Tasks. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, Amar Phanishayee and Vyas Sekar (Eds.). USENIX Association, 1–18. https://www.usenix.org/conference/nsdi22/presentation/mcclure

[25] Gal Mendelson, Shay Vargaftik, Katherine Barabash, Dean H. Lorenz, Isaac Keslassy, and Ariel Orda. 2021. AnchorHash: A Scalable Consistent Hash. *IEEE/ACM Trans. Netw.* 29, 2 (2021), 517–528. https://doi.org/10.1109/TNET.2020.3039547

[26] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 115–130. https://doi.org/10.1145/3318464.3389758

[27] Djob Mvondo, Mathieu Bacou, Kevin Nguetchouang, Lucien Ngale, Stéphane Pouget, Josiane Kouam, Renaud Lachaize, Jinho Hwang, Tim Wood, Daniel Hagimont, Noël De Palma, Bernabé Batchakui, and Alain Tchana. 2021. OFC: an opportunistic caching system for FaaS platforms. In *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*, Antonio Barbalace, Pramod Bhatotia, Lorenzo Alvisi, and Cristian Cadar (Eds.). ACM, 228–244. https://doi.org/10.1145/3447786.3456239

[28] Yuichi Nakatani. 2021. Structured Allocation-Based Consistent Hashing With Improved Balancing for Cloud Infrastructure. *IEEE Trans. Parallel Distributed Syst.* 32, 9 (2021), 2248–2261. https://doi.org/10.1109/TPDS.2021.3058963

[29] Vladimir Andrei Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. 2018. Stateless Datacenter Load-balancing with Beamer. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, Sujata Banerjee and Srinivasan Seshan (Eds.). USENIX Association, 125–139. https://www.usenix.org/conference/nsdi18/presentation/olteanu

[30] Matthew Perron, Raul Castro Fernandez, David J. DeWitt, and Samuel Madden. 2020. Starling: A Scalable Query Engine on Cloud Functions. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 131–141. https://doi.org/10.1145/3318464.3380609

[31] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 193–206. https://www.usenix.org/conference/nsdi19/presentation/pu

[32] An Qin, Yuan Yuan, Dai Tan, Pengyu Sun, Xiang Zhang, Hao Cao, Rubao Lee, and Xiaodong Zhang. 2017. Feisu: Fast Query Execution over Heterogeneous Data Sources on Large-Scale Clusters. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 1173–1182. https://doi.org/10.1109/ICDE.2017.162

[33] Francisco Romero, Gohar Irfan Chaudhry, Iñigo Goiri, Pragna Gopa, Paul Batum, Neeraja J. Yadwadkar, Rodrigo Fonseca, Christos Kozyrakis, and Ricardo Bianchini. 2021. Faa$T: A Transparent Auto-Scaling Cache for Serverless Applications. In *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, Carlo Curino, Georgia Koutrika, and Ravi Netravali (Eds.). ACM, 122–137. https://doi.org/10.1145/3472883.3486974

[34] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2021. Characterizing and Mitigating the I/O Scalability Challenges for Serverless Applications. In *IEEE International Symposium on Workload Characterization, IISWC 2021, Storrs, CT, USA, November 7-9, 2021*. IEEE, 74–86. https://doi.org/10.1109/IISWC53511.2021.00018

[35] Alexander Slesarev, Mikhail Mikhailov, and George Chernishev. 2022. Benchmarking Hashing Algorithms for Load Balancing in a Distributed Database Environment. In *Advances in Model and Data Engineering in the Digitalization Era*, Philippe Fournier-Viger, Ahmed Hassan, Ladjel Bellatreche, Ahmed Awad, Abderrahim Ait Wakrime, Yassine Ouhammou, and Idir Ait Sadoune (Eds.). Springer Nature Switzerland, Cham, 105–118.

[36] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. 2020. Cloudburst: Stateful Functions-as-a-Service. *Proc. VLDB Endow.* 13, 11 (2020), 2438–2452. http://www.vldb.org/pvldb/vol13/p2438-sreekanti.pdf

[37] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 27-31, 2001, San Diego, CA, USA*, Rene L. Cruz and George Varghese (Eds.). ACM, 149–160. https://doi.org/10.1145/383059.383071

[38] David Thaler and Chinya V. Ravishankar. 1998. Using name-based mappings to increase hit rates. *IEEE/ACM Trans. Netw.* 6, 1 (1998), 1–14. https://doi.org/10.1109/90.663936

[39] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 449–462. https://www.usenix.org/conference/nsdi20/presentation/vuppalapati

[40] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua

Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2614–2627. https://doi.org/10.1145/3448016.3457550

A USING THE ARTIFACTS

The artifact is available on Zenodo at https://zenodo.org/record/7812238.

The repository contains the data, the simulator, and the support scripts necessary to reproduce the plots from our paper.

We use data from the IBM COS [14] dataset, but extract and subset of it and convert it into a format suitable for simulation. These simulable traces are in the *input_traces1* folder.

We add our distributed cache simulator as a component to OpenDC [23] datacenter simulation suite. The specific version of the source code we use in this work can be found at https://github.com/sacheendra/opendc/tree/cachearch. We include a pre-compiled jar file of the simulator in this repository.

The three support scripts enumerated 1, 2, and 3 use the Ray distributed programming framework to run multiple experiments simultaneously. Each simulation requires a dedicated CPU core and 10GB of RAM. On a single server with 20 cores, all the simulations complete in about half an hour. Running the simulations on a cluster requires cluster-specific setup for Ray.

A.1 Requirements

(1) Linux-based OS
(2) JDK 17 (for the simulator)
(3) Python 3.11
(4) A LaTeX distribution (for the plots)

A.2 Building the Simulator

This repository comes with a pre-built version of the simulator. But, this is the procedure if a user chooses to build their own.

(1) Clone the latest source code from https://github.com/sacheendra/opendc/tree/cachearch.
(2) Build the simulator using the command *./gradlew :opendc-storage:opendc-distributed-cache:fatJar*.
(3) The jar file should will be in the *opendc-storage/opendc-distributed-cache/build/libs* folder.
(4) Copy the jar file *opendc-distributed-cache-3.0-SNAPSHOT.jar* to this repository.

A.3 Running the Simulations

(1) Create a python virtual environment using a tool of your choice. Conda and virtualenv both work.
(2) Use python 3.11.
(3) Install the required libraries using *pip -r requirements.txt*.
(4) Run the simulations using *python '1. run_simulations.py'*.
(5) Next, summarize the results using *python '2. summarize_results.py'*.
(6) The plots will be in the *result_plots3* folder.