Performance Evaluation of Data-Centric Workloads in Serverless Environments

Anna Maria Nestorov
Barcelona Supercomputing Center
Universitat Politècnica de Catalunya
anna.nestorov@bsc.es

Jordà Polo
Barcelona Supercomputing Center
jorda.polo@bsc.es

Claudia Misale
IBM T.J. Watson Research Center
c.misale@ibm.com

David Carrera
Barcelona Supercomputing Center
david.carrera@bsc.es

Alaa S. Youssef
IBM T.J. Watson Research Center
asyousse@us.ibm.com

***Abstract*—Serverless computing is a cloud-based execution paradigm that allows provisioning resources on-demand, freeing developers from infrastructure management and operational concerns. It typically involves deploying workloads as stateless functions that take no resources when not in use, and is meant to scale transparently. To make serverless effective, providers impose limits on a per-function level, such as maximum duration, fixed amount of memory, and no persistent local storage. These constraints make it challenging for data-intensive workloads to take advantage of serverless because they lead to sharing significant amounts of data through remote storage.**

In this paper, we build a performance model for serverless workloads that considers how data is shared between functions, including the amount of data and the underlying technology that is being used. The model's accuracy is assessed by running a real workload in a cluster using Knative, a state-of-the-art serverless environment, showing a relative error of 5.52%. With the proposed model, we evaluate the performance of data-intensive workloads in serverless, analyzing parallelism, scalability, resource requirements, and scheduling policies. We also explore possible solutions for the data-sharing problem, like using local memory and storage. Our results show that the performance of data-intensive workloads in serverless can be up to 4.32× faster depending on how these are deployed.

***Index Terms*—Serverless, Distributed Computing, Kubernetes, Knative, Tekton, Performance Model, Storage**

I. Introduction

In recent years, *serverless computing* has received a significant uptick in attention. In this event-driven paradigm, the operational concerns of managing the infrastructure are entirely left to the cloud provider, letting more users approach the cloud and allowing them to be completely focused on their domain of expertise. Its fundamental principles, such as 'pay-as-you-go' billing and auto-scaling all the way down to zero resources in the absence of requests, make serverless attractive to many users.

Serverless has shortfalls and presents challenges, as depicted in the literature [1–5]. Due to the stateless nature of serverless, only a subset of workloads, e.g., web microservices and IoT applications, currently benefit from it. In the context of data-intensive workloads, characterized by large resource requirements and significant amounts of data transfers, none of the major serverless platforms, such as AWS Lambda, Google Cloud Functions, IBM Cloud Functions, and Microsoft Azure Functions, offer efficient support. The imposed memory and storage limitations at function level represent a major constraint. For example, in AWS Lambda, each function can only allocate from 128MB to 3GB of memory and a maximum of 512MB of ephemeral storage. Another significant constraint of public cloud solutions is the limitation to share data only through remote shared storage, completely cutting off data locality. Hence, directly using a serverless platform for data-intensive workloads could lead to extremely inefficient executions [6].

While the desirability and advantages of enabling more complex workloads in serverless environments are known, its feasibility and efficiency still remain unclear. There is a need to fully understand 1) data exchange mechanisms between functions in current serverless platforms, and whether new approaches are needed, and 2) data flows of these workloads, considering their parallelism and resource requirements. Aiming to understand the possible data exchange solutions and the impact of complex data flows in serverless environments, the main contributions of this paper are:

- A performance model capturing the performance of serverless workloads with data exchanges, supporting different configurations and models of exchanging data.
- A complete characterization of a data-intensive workload, showing a methodology to extract the necessary components to translate a traditional application to serverless.
- A comprehensive evaluation of data-intensive workloads in serverless environments, exploring parallelism, data exchange, resource allocation, and scheduling policies.

This paper is organized as follows. Section II introduces the reader to the Kubernetes, Knative, and Tekton platforms, and summarizes related work. Section III describes the proposed performance model. Section IV presents the use case and validation workload, along with its characterization. Section V reports the experimental setup and results. Finally, Section VII draws the conclusions and future work.

II. Background and Related Work

While there are a number of available container orchestration frameworks, Kubernetes [7] has become the leading platform and de-facto standard. Kubernetes is an open-source platform for automating deployment, scaling, and management of containerized applications across multiple cloud providers platforms, as well as on dedicated Virtual Machines (VMs), or on bare metal. Knative [8] introduces event-driven and serverless capabilities for Kubernetes clusters and has become the open serverless standard. In particular, it manages stateless services by reducing the users' effort required for autoscaling, networking, and rollouts, and it routes events between on-cluster and off-cluster components. Tekton [9] represents a powerful yet flexible Kubernetes-native framework for designing and running event-driven and serverless workflows.

A big challenge when deploying data-intensive workloads on serverless computing platforms is efficiently sharing data between tasks. Several works in the literature have proposed relaxing storage constraints to favor performance: [10, 11] share the idea of co-locating code with the data it accesses.

There have been a few proposals, built on top of AWS Lambda, to orchestrate distributed computing running on serverless focusing on storage [6, 12]. In [6], the authors propose Locus, a serverless model for the most general all-to-all shuffle scenario that combines cheap but slow storage with fast but expensive storage, and predicts shuffle performance with an average error of 15.9%. Unlike Locus, our aim with the model presented in this paper is more general and should apply to any kind of workload. [12] highlights the relevance of efficient data communication between functions via a shared data store in analytics workloads. The work analyzes three different storage systems, i.e., a disk-based managed object storage service, an in-memory key-value store, and a flash-based distributed storage system. In particular, the authors show that S3 has significant overhead, particularly for small requests. In this work, we only account for fully-managed storage solutions, as in-memory key-value stores and flash-based storage would require the user to select instance types, or to manually configure and scale the resources.

III. Performance model

The goal of the proposed model is to estimate the end-to-end performance of a non-trivial workload with data dependencies between the pipeline tasks, as if it was running in a datacenter on top of a Knative-based serverless environment. Since there is a significant emphasis on data, the model allows for different exchange mechanisms, including local memory, local storage, and remote storage. While some of these may not be supported or are limited in current serverless implementations, they can still be used to evaluate alternative solutions.

The inputs of the model are: 1) hardware platform description, 2) scheduling strategy, 3) workload input, 4) workload graph, and optionally 5) bandwidth characterization. The hardware platform description (1) input provides the necessary information about the cluster machines' memory and storage. For memory, it specifies the data width and the speed, necessary to compute the theoretical memory bandwidth, while for storage, the maximum transfer rate is reported. The scheduling strategy (2) input specifies the task-node assignments. The workload (3) input defines the input dimension, the total execution time, and the amount of data it contains. The workload graph (4) input specifies for each task its name, predecessor tasks names, inputs/outputs object names, compute time or profiling real percentage, parallelisms object with their possible unrolling factors. With *unrolling factor* we denote the number of concurrent task executions for a specific parallelism. Finally, the optional bandwidth characterization (5) input provides, for a set of data dimensions, the median read/write bandwidths for the three accounted data exachange mechanisms. If this last input is provided, the model returns a more realistic estimation; otherwise, it provides a best-case prediction by using the theoretical bandwidths.

Given a pipeline composed of *n* tasks, each *i*-th task total time is composed of: the time necessary to read the input data T_i^r, the compute time T_i^c, and the time necessary to write the output data T_i^w. Given the *i*-th task exploits *m* parallelisms, for each *j*-th parallelism, we denote with $x_{i,j}$ the number of sequential computations in case of no task replication and with $p_{i,j}$ the unrolling factor. Thus, $\left\lceil \frac{x_{i,j}}{p_{i,j}} \right\rceil$ represents the actual number of computations the task has to perform. In case of no concurrent execution, the model considers $p_{i,j} = 1$. We derive the end-to-end pipeline time T_{tot} as:

$$T_{tot} = \sum_{i=1}^{n} \left[\left(\prod_{j=1}^{m} \left\lceil \frac{x_{i,j}}{p_{i,j}} \right\rceil \right) (T_i^r + T_i^c + T_i^w) \right] \tag{1}$$

where:

$$T_i^r = \sum_{k=1}^{3} \left(\sum_{s=0}^{ni} \frac{D_s}{BW_k^r(D_s)} \right) \tag{2}$$

$$T_i^w = \sum_{k=1}^{3} \left(\sum_{u=0}^{no} \frac{D_u}{BW_k^w(D_u)} \right) \tag{3}$$

As shown in Equation (2) and Equation (3), to account for the three data exchange mechanisms, the model computes the read time T_i^r and the write time T_i^w as the sum of the *ni* inputs and *no* outputs partial communication latencies, computed by dividing the amount of data to be read/written *D* by the specific mechanism bandwidth *BW*. Bandwidths *BW* are functions of the amount of data *D*. If the bandwidth characterization input is given and the target data dimension D is not present in the set of benchmarked dimensions, the bandwidth BW is estimated by applying the linear interpolation method.

IV. Use Case and Validation Workload: Tesseract

To validate the proposed performance model we characterize and study Google's Tesseract Optical Character Recognition (OCR) engine [13], representing the current open source state-of-the-art. It is an interesting use case study because it represents a more traditional workload with non-trivial computational and data graphs, not currently well supported in serverless environments. Tesseract features smaller data transfers compared to more I/O-intensive workloads; however, it involves a significant amount of data at scale. For example, financial institutions processing large amounts of documents.

A. Methodology

Transforming a traditional application like Tesseract into a serverless workload requires understanding its internal computational graph, and splitting the application into smaller computational units. Obtaining the computational graph involves profiling the application to extract a detailed *call-graph* representing the workload call chains using tools like the Valgrind profiler along with Callgrind [14].

Since the code base of applications like Tesseract can be very large, it would not be feasible to map every application function into an independent serverless function. Hence we simplify the call-graph by grouping functions into computational units called *macro-nodes*. These macro-nodes can be further evaluated by instrumenting the application code, and using memory-profiling tools like Massif [14]. Since such tools do not provide any information on the correlation between memory allocations and variables/objects to which the memory belongs, we build a set of custom tools to trace the average dimension of each macro-node's inputs and outputs.

Finally, to highlight data exchanges, we create a *dataflow* graph where nodes represent macro-nodes and arcs represent data channels. Macro-nodes with small cost-times and trivial data movements are merged with neighboring macro-nodes.

B. Tesseract Characterization

To characterize Tesseract, we evaluate the execution time and the data dimensions of two single-page documents, *DataSetA* and *DataSetB*, that present different page structure and number of lines. To evaluate the scalability, datasets are replicated generating increasingly bigger inputs, from 1 to 128 pages. All the experiments are executed on an Intel Xeon Silver 4114 CPU running at 2.20GHz.

Table I presents the simplified Tesseract call-graph, where the macro-node real percentage is computed by subtracting the children inclusive percentages from its inclusive value. The three main Tesseract phases, namely pre-processing, processing and post-processing, are identified by the *FindLines*, *RecogAllWords* and *DetectParagraph* macro-nodes, taking 12.54%, 83.82%, and 1.17% over the total execution time, respectively. The processing phase is the most time-consuming, mainly due to the Long Short Term Memory (LSTM) based Neural Network (NN). The workload is characterized by three parallelisms: at page, block, and textline-level.

In experiments not shown for space limitations, we find that the TIFF conversion time scales linearly with the number of pages and is strongly dependent on the page dimension. Specifically, a single PDF page in A4 format takes 1.19s to be converted. Concerning Tesseract computation, the single-threaded implementation average execution times amount to 9.06s for DataSetA and 11.41s for DataSetB. The corresponding multi-threaded implementations are only 15% faster on average, since threads are only used for some NN computations. We observe that both single and multi-threaded implementations scale linearly with the number of pages.

TABLE I: Tesseract simplified call-graph

Macro-node	**Percentage [%]**		**Parallelism**	**Granularity**	
	Inclusive	**Real**		**CG**	**FG**
main	99.97	0.28	-	1	1
ProcessMultipageTIFF	99.69	1.56	-	1	1
Recognize	98.13	0.60	Page	2	2
FindLines	12.54	0.01	Page	2	2
ExtractThresholds	2.33	2.33	Page	2	2
SegmentPage	10.20	0.01	Page	2	3
AutoSegmentation	8.13	0.01	Page	2	3
SetupSegCorOrient	4.18	4.18	Page	2	3
FindBlocks	3.94	3.94	Page	2	4
MakeTextlinesWords	2.06	2.06	Page, Block	2	5
RecogAllWords	83.82	0.20	Page	3	6
LSTMRecogWords	83.62	0.01	Page, Textline	3	7
GetLineImage	2.60	2.60	Page, Textline	3	7
RecognizeLine	81.01	0.04	Page, Textline	3	8
LSTM-NN	79.65	79.65	Page, Textline	3	8
DecodeNNOutput	1.32	1.32	Page, Textline	3	9
DetectParagraphs	1.17	1.17	Page	4	10

Last, we analyze the workload memory footprint and we highlight the input and output dimensions of the initial conversion step and of each workload macro-node. The memory footprint is up to 150MiB, with a short peak of 166.9MiB, in the first two-thirds of the workload execution and up to 90MiB in the last one-third. For the initial conversion step, a page of roughly 40KB is converted into a 8.7MB image, and grows linearly with the number of pages, so a 128 pages input takes 1.09GB. By analyzing the dataflow graph, we observe that data movements are in the order of tens of MB. While this seems relatively small, data transfers could become prohibitive when exploiting parallelism.

To evaluate Tesseract in serverless environments, in this paper we provide two scenarios with different task *granularity*, which defines at what level code functions are grouped to become serverless tasks. Accounting with the initial TIFF conversion step, the **Coarse-Grained (CG)** scenario consists of *bigger* groups for a total of 5 tasks, while the **Fine-Grained (FG)** scenario consists of *smaller* groups for a total of 11 tasks. Table I shows the mapping between macro-nodes and tasks of the two different scenarios.

V. Evaluation of the Model

In this section, we conduct an accuracy analysis of the proposed performance model. Experiments are executed on a virtualized Kubernetes cluster composed of one master, representing the control-plane, and three worker nodes on which tasks are deployed, along with an Network File System (NFS) server inside the same infrastructure. The master runs in a VM with 32GB of memory and 16 virtual cores, while the three workers and the NFS server run in a VM with 32GB of memory, 8 virtual cores, and 50GB of disk space each. The Kubernetes cluster and the NFS server are mapped on five physical nodes residing in the same rack and featuring an Intel Xeon Silver 4114 CPU running at 2.20GHz connected to four

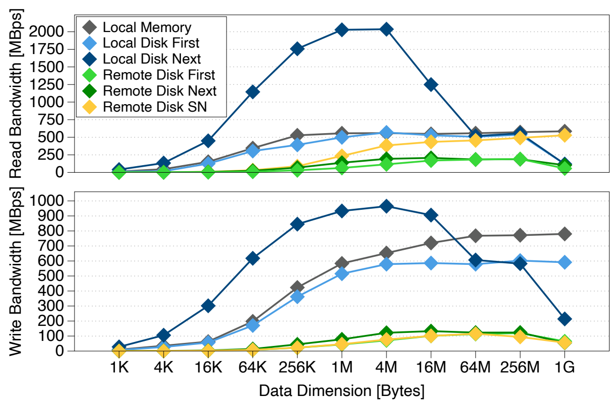

Fig. 1: Read and write bandwidths. First and Next keywords highligh the first and subsequent local/remote disk bandwidths, while Remote Disk SN refers to remote operations with producer and consumer co-placed on the same node.

Seagate SATA HDDs. Physical nodes are connected through a 10Gbps network switch. The memory and the disk feature 17GBps and 175MBps of maximum transfer rate, respectively.

A. *Model: Bandwidth Characterization*

One of the model inputs is the characterization of the different communication mechanisms, i.e., local memory, local storage, and remote storage. In this characterization we want to capture the performance of the same complete stack that can be found in a serverless platform, and not just the performance of the devices. The experiments are based on two kinds of tasks: *producer* tasks writing data, and *consumer* tasks reading data. Between producer-consumer task pairs, we expect data to be accessed only sequentially. Focusing on storage solutions inside the same infrastructure, data can be shared by either using a local Persistent Volume (PV) (local memory and storage), or a remote PV map on the NFS server (remote storage). The latter can also be used to simulate an environment similar to that of major public serverless providers. We evaluate data exchanges of different sizes, ranging from 1KB to 1GB with a $4\times$ increment.

Figure 1 shows the sequential bandwidths of the considered data exchange mechanisms. For both read and write operations, the local memory and the local disk (First) solutions exhibit similar performance. Thanks to the *disk buffering* computing infrastructure optimization, allowing to buffer data in main memory *buffer cache* space, the local disk read and write bandwidths reach a maximum of 535MBps and 602MBps, which are much higher than the 175MBps disk maximum transfer rate reported by the vendor. Differently, the remote data requests are handled by communicating directly with the disk. The only exception is when two tasks are co-placed: the producer task writes the data on the remote disk, while the consumer task reads from the memory buffer cache. We notice that in both local and remote solutions, when a task reads/writes multiple times sequentially on the same PV, e.g., a task collecting multiple data in input from various producers, the first operation and the following ones exhibit different bandwidths. More precisely, the subsequent local and remote read and write bandwidths are up to $4.41\times$ and $2.14\times$, and $3.34\times$ and $1.88\times$ higher when compared to the first operations bandwidths, accordingly. The local disk bandwidth increase is due to caching, while, being the data read/written sequentially, the subsequent remote operations take advantage of the sequential locality at the hard disk level.

TABLE II: Model accuracy.

	KPI	Coarse-grained		Fine-grained	
		1 page	**10 pages**	**1 page**	**10 pages**
Local	**Avg I/O Time [s]**	0.324	2.890	0.670	7.362
	Pred I/O Time [s]	0.319	2.753	0.750	7.055
	Avg Rel Error [%]	3.09	4.21	7.70	5.64
Remote	**Avg I/O Time [s]**	1.189	9.303	5.388	49.938
	Pred I/O Time [s]	1.099	9.283	5.579	53.813
	Avg Rel Error [%]	6.90	3.64	5.61	7.42

Since the model is meant to handle many tasks running and sharing data at the same time, we also evaluate the performance degradation with concurrent requests. We characterize concurrent read and write bandwidths by manually enforcing the synchronization of the executed tasks. As expected, by increasing the number of concurrent requests, the single task bandwidth decreases, especially with high data dimensions. More specifically, with 30 concurrent tasks, the read and write bandwidth experience the highest performance degradation with a decrease of up to $4.60\times$ and $5.98\times$, and to $26.72\times$ and $38.55\times$, for local and remote operations, accordingly.

B. *Model: Accuracy*

To validate and measure the accuracy of the proposed model, we compute the average relative error of the inter-function I/O time prediction with respect to the actual communication overhead of 100 Tesseract-like deployment runs over two different input datasets, featuring 1 and 10 pages. Following a classical black-box approach, with *Tesseract-like* deployment we mean a deployment where each task performs the same data exchanges described in IV-B, and sleeps during the specified time. To be fairer in the results, since compute time is constant and significantly larger than inter-function I/O time, we consider only the I/O time in the measurements. We run the Tesseract-like computations in two configurations: the first co-places all the tasks on a single node following the default Knative/Tekton strategy, while the second maps the tasks on the worker nodes in a round-robin fashion. These two configurations are referred to as *local* and *remote*.

To validate our model, we only consider sequential executions, representative of several real-world serverless applications [15, 16]. We run exhaustive experiments with varying data dimensions and number of operations, and compare the I/O times with the model predicted values. In particular, we run the coarse-grained and fine-grained Tesseract-like workloads exploiting both local and remote storage configurations. As shown in Table II, our model predicts communication latency with an average relative error of 5.52%, which we believe is accurate enough to analyze the I/O time impact when porting a workload to Knative.

VI. Evaluation of Data-aware Serverless

Thanks to the flexibility of the proposed model, we investigate the performance impact on the use case workload of the following key factors: task granularity and concurrency, data locality, resource allocation, and scheduling policies.

A. Task Granularity and Concurrency

Task granularity and concurrency are two relevant factors that could highly increase or reduce performance in terms of number of deployed task-instances and overall end-to-end time. Finding the optimal task granularity and concurrency becomes crucial when moving to serverless.

Figure 2 shows the effect of different granularities and concurrencies on the number of deployed task-instances and the end-to-end time in local and remote deployments on a ten pages input with 100 text lines each. While the coarse-grained (CG) solution exploits page-level parallelism, the fine-grained (FG) solution exploits page-level and textline-level parallelism up to an unrolling factor of 25. Block and textline-level parallelism with greater unrolling factors are not considered since they negligibly impact performance. The fully unrolled coarse-grained deployment increases the number of task-instances from 5 to 32 and achieves an end-to-end time speedup of $4.57\times$ and $3.44\times$, for local and remote deployments, respectively. The fully unrolled fine-grained deployment significantly increases the number of task-instances from 11 to 812, achieving a speedup of $6.79\times$ and $4.24\times$, accordingly. For similar configurations (depicted as circles in Figure 2), coarse-grained deployment outperforms fine-grained deployment, especially in the remote configuration where it achieves up to $1.55\times$ of performance increase. On the other hand, fine-grained executions achieve higher speedups, but at the expense of number of tasks and I/O. This needs to be considered because the number of tasks may be limited in local configurations, and I/O may become a bottleneck in remote configurations. In the remote configuration, the network load changes substantially: with fully-unrolled coarse-grained deployment it reaches 1.4GB, while it grows up to 15.6GB with fully-unrolled fine-grained deployment.

B. Data Locality

The local deployment outperforms its equivalent remote deployment in both coarse-grained and fine-grained scenarios, as shown in Figure 2. However, fully local deployments are not realistic for many applications, particularly in large clusters where remote deployments may be more flexible and improve overall throughput and resource usage.

A shortcoming of existing serverless frameworks is that tasks that are co-placed on the same node and share the same inter-function data (e.g. functions that read the same input) do not currently benefit from data reuse. That is, each instance of a function on the same node will fetch the same inter-function input data. Enabling data-intensive serverless workloads will likely require some optimizations in this regard. Creating a local memory or disk buffer holding the shared input data might be a simple way to benefit from data locality, decreasing requests for remote data, and improving its performance.

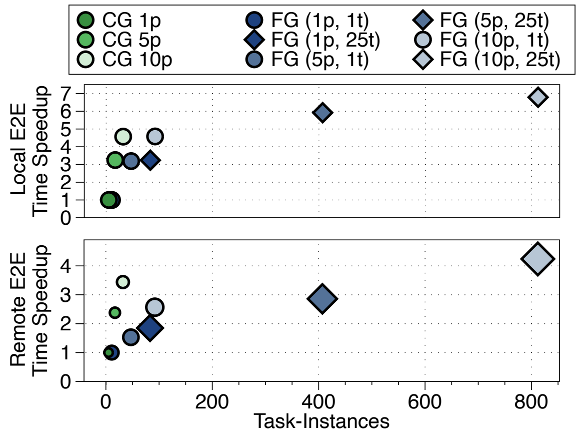

Fig. 2: Local and remote end-to-end time speedups versus the sequential execution, achieved by exploiting page and textline-level parallelisms (p, t), and task-instances number. The remote deployment reports network load impact with increasingly bigger symbol sizes.

C. Resource Allocation

In an ideal serverless environment, the user should not need to provide any resource configuration. However, existing frameworks still allow customizing expected resource usage for improved performance and cost. With the default resource allocation, tasks under-utilizing the assigned resources could prevent other tasks from being scheduled, leading to worse overall system throughput. This is particularly relevant for many data-intensive workloads since tasks belonging to the same pipeline may have significantly different resource requirements, and these can also change depending on the input.

For example, focusing exclusively on memory for the sake of simplicity, we get approximately 31GiB of allocatable memory in each one of the worker nodes in the evaluation environment. In a coarse-grained scenario, and allocating the same peak memory of 167MiB (see Section IV-B) to all 5 tasks, we would be able to fit up to 38 Tesseract deployments per node. With a more accurate allocation of 150MiB, 167MiB, 150MiB, 90MiB, 90MiB for each one of the five tasks, we would be able to fit up to 49 Tesseract deployments.

D. Scheduling Policies

Most of the schedulers proposed in the literature consider properties such as CPU usage, memory utilization, job execution time, and job deadline. Another essential factor to consider when optimizing data-intensive task placement is the effect on the network. By leveraging on the proposed model, we highlight the primary importance of data locality in reducing communication overheads by showing the impact on the overall end-to-end time of four different scheduling strategies: (1) random, (2) round-robin, (3) consolidating, and (4) data-centric. The first two are self-explanatory. The *consolidating* strategy places tasks at per-node level in a sequential fashion, saturating the resources available for each node. The *data-centric* strategy aims to reduce inter-node data movements, and considers the number and amount of data to be shared and

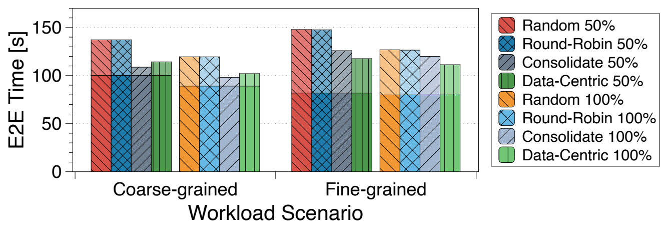

Fig. 3: Comparison of scheduling policies running a 50 page input on a cluster with 100 nodes. Horizontal bars divide computational time (bottom) and inter-function I/O time (top).

exchanged between functions. For the use case workload, the data-centric strategy maps for the coarse-grained scenario the *processMultipageTIFF*, *Pre-Processing* and *Processing* tasks on the same node. For the fine-grained scenario, instead, the set of tasks computing the same page and belonging to the pre-processing and processing phase are co-placed, while spreading the remaining ones randomly. This allows to reduce the communication overhead by exploiting the higher local bandwidths for the most relevant data movements, and to reduce the number of remote data exchange.

Figure 3 shows the performance achieved by the four strategies when executing a 50 pages input on a 100 nodes cluster, with unrolling factors of 50% and 100%. It is noticeable that the random and round-robin performance are similar, and that they are outperformed by both the consolidating and data-centric strategies. In particular, the consolidating strategy achieves an I/O time speedup of 4.32$\times$ and 3.40$\times$ in the coarse-grained scenario, and of 1.49$\times$ and 1.16$\times$ in the fine-grained scenario, for 50% and 100% unrolling factors, respectively. The data-centric solution gains an I/O time speedup of 2.62$\times$ and 2.35$\times$ in the coarse-grained scenario, and of 1.84$\times$ and 1.48$\times$ in the fine-grained scenario, for 50% and 100% unrolling factors accordingly. The coarse-grained workload deployment is composed of 77 and 152 tasks for 50% and 100% unrolling factors, respectively; its consolidating placement performs better than the data-centric solution because it relies on fewer nodes, i.e., one or two. Differently, the fine-grained workload deployment comprises a much higher number of tasks, i.e., 2027 and 4052; therefore, the data-centric solution impact on the network load is lower than the consolidating solution. More specifically, the data-centric achieves a 1.23$\times$ and 1.27$\times$ of performance improvement versus the consolidating strategy, for 50% and 100% unrolling factors, respectively. While the consolidating solution sequentially maps tasks, the data-centric one efficiently places all the tasks characterized by the page-level parallelism as a unit, reducing the communications over the network.

VII. Conclusions

With the introduction of serverless computing, there is a growing interest in adopting these environments for different kinds of workloads, including data-intensive pipelines. In this paper, we propose a performance model for serverless workloads that accurately predicts the performance of inter-function data exchanges with an average error of 5.52%. We also analyze the performance of data-intensive workloads in terms of parallelism, data locality, resource requirements, and scheduling policies. Depending on how workloads are deployed and scheduled, our evaluation shows that performance can be increased up to 4.32$\times$.

As future steps, we believe that more effort is needed to automate the configuration of task granularity and concurrency to take advantage of the different tradeoffs between performance and resource usage, and also to explore more scheduling strategies to optimize data-intensive workloads in serverless environments.

Acknowledgments

This work was partially supported by the Ministry of Economy of Spain under contract TIN2015-65316-P, the Ministry of Science under contract PID2019-107255GB-C21/AEI/10.13039/501100011033, and the Generalitat de Catalunya under contract 2014SGR1051.

References

[1] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and function-as-a-service(faas) in industry and research," *CoRR*, vol. abs/1708.08028, 2017. arXiv: 1708.08028.
[2] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 2018, pp. 159–169.
[3] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. Popa, I. Stoica, and D. Patterson, *Cloud programming simplified: A berkeley view on serverless computing*, Feb. 2019.
[4] P. G. López, M. S. Artigas, S. Shillaker, P. R. Pietzuch, D. Breitgand, G. Vernik, P. Sutra, T. Tarrant, and A. J. Ferrer, "Servermix: Tradeoffs and challenges of serverless data analytics," *CoRR*, vol. abs/1907.11465, 2019. arXiv: 1907.11465.
[5] J. M. Hellerstein, J. M. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," *CoRR*, vol. abs/1812.03651, 2018. arXiv: 1812.03651.
[6] Q. Pu, S. Venkataraman, and I. Stoica, "Shuffling, fast and slow: Scalable analytics on serverless infrastructure," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA: USENIX Association, Feb. 2019, pp. 193–206.
[7] Kubernetes (K8s), [Online]. Available at: https://kubernetes.io.
[8] Knative, [Online]. Available at: https://knative.dev.
[9] Tekton Cloud Native CI/CD, [Online]. Available at: https://tekton.dev.
[10] Z. Al-Ali, S. Goodarzy, E. Hunter, S. Ha, R. Han, E. Keller, and E. Rozner, "Making serverless computing more serverless," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 456–459.
[11] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, *Serverless computing: One step forward, two steps back*, 2018. arXiv: 1812.03651 [cs.DC].
[12] A. Klimovic, Y. Wang, C. Kozyrakis, P. Stuedi, J. Pfefferle, and A. Trivedi, "Understanding ephemeral storage for serverless analytics," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA: USENIX Association, Jul. 2018, pp. 789–794.
[13] R. Smith, "An overview of the tesseract ocr engine," in *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ser. ICDAR '07, USA: IEEE Computer Society, 2007, pp. 629–633.
[14] Valgrind, [Online]. Available at: https://valgrind.org/info/tools.html.
[15] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19, Dresden, Germany: Association for Computing Machinery, 2019.
[16] T. Elgamal, A. Sandur, K. Nahrstedt, and G. Agha, *Costless: Optimizing cost of serverless computing through function fusion and placement*, 2018. arXiv: 1811.09721 [cs.DC].