**MSc Projects by Alexandru Uta, assistant professor in the systems group at LIACS.**

Alexandru's area of expertise is in large-scale computer systems, distributed data processing systems, cloud computing, and performance evaluation and benchmarking.

The following projects represent low-level (distributed) systems work, with a clear focus on *performance evaluation*, *large-scale experimentation,* and *system design and implementation.* By taking on any of these projects you will learn in-depth about all three aspects enumerated above. This will prepare you for either industry and academia. The skills needed to tackle these projects are practical and theoretical knowledge on operating systems, networks, parallel (and distributed) programming, and general software engineering. All projects will deliver open-source code and open-data artifacts.

1. **Performance variability testbed**

In clouds, the performance of individual resources (e.g., CPU, memory, disk, network) is highly variable. This means that running an application at different points in time, there is a high probability that the level of performance will be (significantly) different. This is either due to interference (e.g., noisy neighbors) or the interaction between the actual client workload and the provider rate limiting policies (e.g., network or CPU token buckets). Therefore, due to variability, running workloads in the cloud attains a questionable amount of confidence and various statistical techniques need to be enforced to achieve credible results [1]. This leads to running too many experiments in the cloud, which can be prohibitively expensive. In this project, we will maintain an inventory of sources of variability and their behavior, and implement them in a *performance variability testbed*, that emulates the behavior of a real cloud in a smaller-scale private infrastructure. Using this instrument, experimenters will be able to determine in advance: (i) the amount of variability their workloads would employ; (ii) root-cause analysis of experiment variability; (iii) how to plan experiments in the cloud.

2. **Spark indexing to (local) SSD or NVMe**

Distributed data processing frameworks, such as Spark, are designed to provide coarse-grained operations on large-scale immutable datasets. This makes it difficult to apply certain optimizations to speed up data processing, such as indexing. We already have a prototype implementation of an in-memory index [2] working with Spark datasets. However, most real-world datasets are larger-than-memory, and we would like to speed the execution of applications analyzing these datasets. In this project, we will implement an out-of-core index that stores data in (local) SSDs or NVMes to allow for increased data sizes while maintaining good performance -- high bandwidth and low latency.

3. **Microservices/serverless testbed**

There are many open-source frameworks and projects supporting serverless and microservices workloads. At the interface with the client, they offer different programming languages and models. At the resource management level they offer different deployment engines, such as Kubernetes, OpenWhisk, Docker Swarm. At the virtualization layer, they offer microVMs (e.g., Amazon Firecracker), containers (e.g., Docker). Evaluating and comparing all combinations is much needed for consumers of microservices in order to find the best solution for their application. In this project, we plan to design and implement a microservices/serverless testbed which combinatorially integrates as many of the existing open-source technologies as possible.

4. **Shielding Microservices from performance variability**

As opposed to the old monoliths, distributed applications developed as microservices are usually expressed as DAGs (directed acyclic graphs) composed of many small, short-lived tasks. Due to the many dependencies of microservice-based applications, individual tasks can only be scheduled in a certain order (i.e., one task waits for another to finish). Performance variations of the underlying computing system can therefore have significant ripple effects on the application level. Intermittent slowdowns at task level will not only slow down subsequent tasks in the same DAG, but heavily interfere with the operation of the entire microservice ecosystem by creating straggler DAGs, effectively increasing tail latencies. In this project we will investigate the feasibility of techniques widely used in data processing systems for dynamic query planning, low-latency scheduling, and straggler mitigation.

## 5. Predictable performance for microservices in datacenters with rate limiters

Datacenter networks use rate limiters to control user-generated load. In recent work, we provided evidence supporting that cloud providers' rate limiters, such as host-based token buckets, affect performance predictability. Although datacenters generally suffer from poor performance predictability and constant change, statistically robust techniques to tame such variability exist. However, there is no panacea for the unpredictable interaction between user-induced load and datacenter rate limiters, which heavily reduces service-level performance predictability. This holds especially true for latency-sensitive workloads, such as serverless [4] and microservices. In this project we explore the uncharted interaction between novel microservice architectures and datacenter rate limiters, and build microservice-based systems whose performance is predictable while keeping the benefits of rate limiters.

## 6. Linux scheduling for serverless workloads

Serverless workloads, such as Function-as-a-Service are becoming increasingly important for many domains. Serverless offers seamless orchestration and scheduling of workloads to machines, very fine grained scalability and billing, and the ability to split monolithic workloads into microservices. This led to a booming market that now reached $7B. Serverless functions are deployed onto Linux servers through micro-VM or container solutions, such as the open-source Amazon Firecracker. Such micro-VMs run in the numbers of thousands per physical Linux server per second, with very high over-subscription rates (of many micro-VMs per physical core). The Linux OS was not designed to handle such bursty and fine-grained workloads, with its core scheduling mechanisms and policies designed for more coarse-grained server-based workloads. In this project we aim to quantify the mismatch of the current Linux scheduler with fine-grained serverless workloads, as well as fine-tuning scheduling policies and mechanisms and developing better scheduling schemes.

## References:

[1] A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeyer, C., Maltzahn, R. Ricci, A. Iosup. "Is big data performance reproducible in modern cloud networks?". In USENIX NSDI 2020.
[2] A. Uta, B. Ghit, A. Dave, P. Boncz. "Low-latency spark queries on updatable data". In SIGMOD 2019.
[3] D. Duplyakin, A. Uta, A. Maricq, R. Ricci, "In datacenter performance, the only constant is change", In IEEE/ACM CCGrid 2020.
[4] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta, A. Iosup. "Serverless is more: From PaaS to present cloud computing." IEEE Internet Computing 2018.