Stage M2 2026

Optimisation de la communication inter-fonctions dans les systèmes serverless

Superviseurs : Alessio Pagliari, Marc Shapiro
Contacts : alessio.pagliari@lip6.fr, marc.shapiro@acm.org
Début : Environ février/mars 2026
Durée : 6 mois
Lieu : LIP6, Equipe DELYS, Campus Pierre et Marie Curie, 4 Place Jussieu, 75005 Paris

Contexte

Le paradigme "serverless" [1] permet d'exécuter des fonctions (petits bouts de code) à la demande, sans gérer de serveurs. Une plateforme lance automatiquement une fonction quand un événement arrive, exécute le code, renvoie un résultat, puis libère les ressources. Ce côté éphémère facilite la montée/descente en charge. Exemples de plateformes: AWS Lambda[1], Google Cloud Functions[2], et des solutions open source déployables localement comme OpenFaaS[3] ou OpenWhisk[4]. Le serverless a montré de bons résultats pour le traitement de données par lots, par exemple de type MapReduce [2]. De plus en plus de cas d'usage exigent toutefois du temps réel (IoT, supervision, e-commerce), où les données arrivent en continu et doivent être traitées sans retard par des pipelines de fonctions interconnectées [3].

Motivation et défis. Pour composer des pipelines de traitement (dataflow [3]), les solutions actuelles passent par des systèmes de messagerie externes (Kafka, files cloud). Cette approche ajoute de la latence (dizaines à centaines de ms par hop), nécessite une infrastructure lourde, et impose des sérialisation/désérialisation coûteuses. Une communication directe entre fonctions (sockets, canaux) réduirait la latence et éliminerait les dépendances externes. Cependant, cela soulève des défis systèmes majeurs que ce stage propose d'explorer.

Objectifs

Après une étude de l'état de l'art, l'objectif principal sera de concevoir et prototyper un système de communication directe entre fonctions serverless. L'étudiant s'attaquera à une ou plusieurs des problématiques système suivantes :

- **Cycle de vie et découverte :** Comment synchroniser le démarrage de ces chaînes de traitement éphémères : demande de connexion à des services encore en cours de démarrage, détection de la terminaison…
- **Scaling et routage :** Dans le cas où les fonctions sont répliquées, comment équilibrer le routage, tout en assurant le maintien de la cohérence du traitement lorsque certaines fonctions sont stateful (i.e. elles maintiennent des données entre les traitements) ?
- **Fiabilité :** Comment garantir le fonctionnement complet de la chaîne si l'on considère que les fonctions ne sont pas fiables ? Quels mécanismes pourrait-on mettre en place pour assurer la détection et le recouvrement de la chaîne ?
- **Performance :** Quelles optimisations système peut-on mettre en place pour limiter l'overhead dû à cette architecture FaaS permettant d'atteindre une latence et des débits compatibles avec les exigences des applications de stream computing (mapping mémoire, algorithmes zéro-copie, placement des threads…) ?

Concrètement, le travail comprendra l'implémentation d'un mécanisme de connexion directe entre les fonctions, avec un orchestrateur qui gère le déploiement et la découverte des applications serverless complexes. Le prototype final devra au moins permettre la communication entre plusieurs fonctions dans un pipeline simple déployé sur une plateforme serverless open source.

Technos : plateformes open source (OpenFaaS/OpenWhisk), extensions système en Go/Scala/C/Rust pour les mécanismes bas niveau, fonctions en Python/Go/Java.

Prérequis

Minimaux :

- Bases des systèmes distribués (cohérence, tolérance aux pannes, partitionnement).
- Programmation système (p. ex. sockets, threads, conteneurs Docker).
- Gestion des ressources système et noyau (mémoire, ordonnancement…).
- À l'aise avec au moins un langage (C/Go/Python/Java).

Souhaités (mais apprises pendant le stage si besoin) :

[1] https://aws.amazon.com/lambda/
[2] https://cloud.google.com/functions
[3] https://www.openfaas.com/
[4] https://openwhisk.apache.org/

- Architectures de streaming (Kafka, Flink).
- Notions de consistent hashing, load balancing, service discovery.

Bibliography

[1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Communications of the ACM*, 2019.

[2] D. Barcelona-Pons, P. Sutra, M. Sánchez-Artigas, G. Par\ís, and P. Garc\ía-López, "Stateful serverless computing with crucial," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2022.

[3] Z. Li, C. Xu, Q. Chen, J. Zhao, C. Chen, and M. Guo, "Dataflower: Exploiting the data-flow paradigm for serverless workflow orchestration," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.