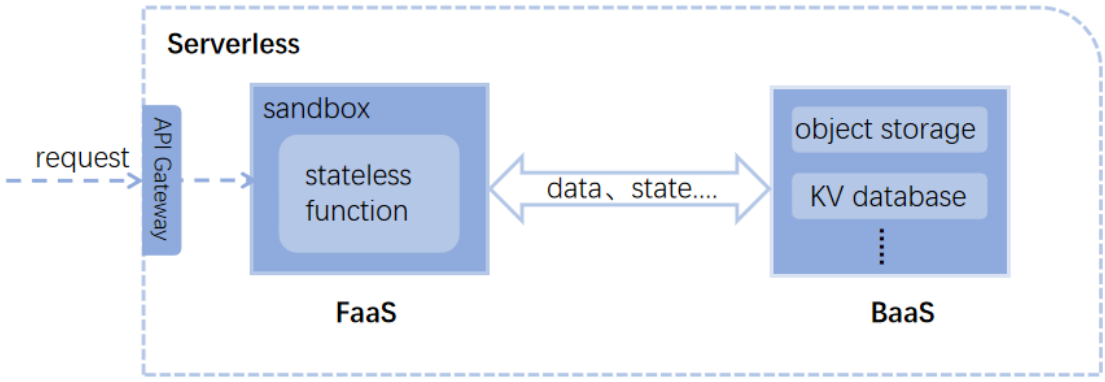
Background

- **Serverless = FaaS + BaaS**

  Serverless采用存储计算分离的方式，由**FaaS**(function as a service)提供计算资源，**BaaS**(backend as a service)提供存储服务。如下图所示：当请求到达时，首先通过API网关路由到对应的沙箱，然后在沙箱中实例化无状态函数进行计算，函数产生的数据和状态或者销毁、或者存储到后端的数据库中。



Serverless computing model

- **Stateless application *vs* Stateful application**

  这里的状态是指存在内存变量中的数据或者写入本地磁盘中的数据。

  - Stateless application
    - 使用本地磁盘来做临时存储，函数执行完临时存储也会清理掉
    - 常见的无状态应用有：Web微服务和IoT应用
    - 通常一个请求只包含一个函数；由于没有保存历史状态，请求复杂；由于使用共享存储保存状态，I/O延迟较高；但是易伸缩
  - Stateful application
    - 本地存储会保存状态
    - 常见的有状态应用有：MapReduce Sort、Query processing
    - 每个请求包含多个阶段，不同阶段的任务需要分享状态和数据；；由于保存历史状态，请求简单；由于使用本地存储存储数据，I/O延迟较低；但伸缩性差

Motivation

- 数据分析应用(Data analytics)，如MapReduce sort、Query processing，通常包含多个处理阶段，每个阶段对资源的需求差异很大，在传统的云服务中按照峰值需求分配资源会造成资源浪费。
- 而如果将Data analytics应用部署在Serverless平台上，由于Serverless高弹性和细粒度收费的特性，可以提高资源利用率，同时降低成本。

Challenge

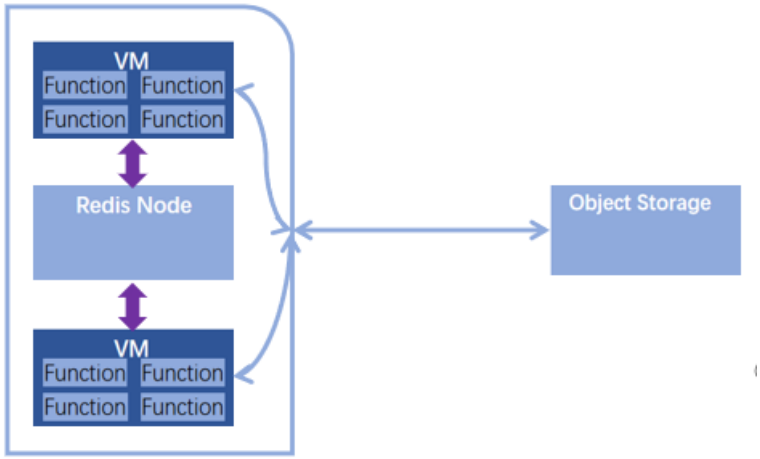
- **Serverless function communication**
  - Serverless平台没有长期运行的应用程序框架代理来管理本地存储，因此无法像Spark等分析框架利用**本地存储**缓存中间数据以实现数据共享
  - 同时Serverless 无服务器应用程序无法控制任务调度或位置，导致Serverless函数无法直接寻址进行直接通信
  - 因此由于以上的限制，Serverless使用远端存储服务，如S3，进行共享数据

- **Data analytics characteristics**
  - 任务阶段间大规模的shuffle操作会导致大量中间数据的读写
- **Challenge**
  - 由于Serverless函数通过远端的缓慢存储共享数据，而Data analytics应用又有大规模小文件的读写，因此会导致较高的I/O时延

Existing Design

External storage

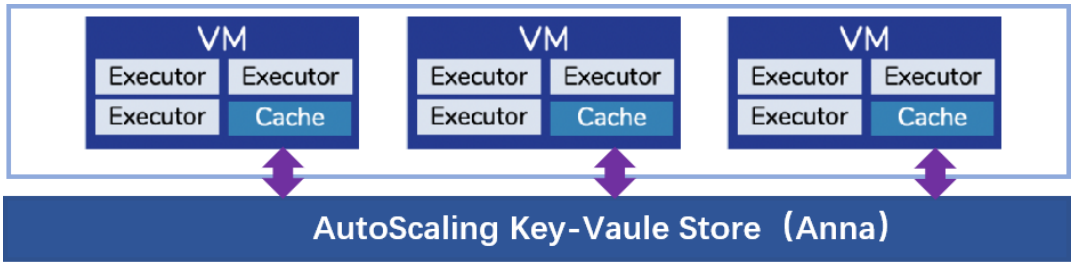
- Locus ***[Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure NSDI'19]***
  - 设计：在FaaS端增加额外的快速存储（memory-based storage）资源节点，然后将任务划分为N轮，每轮中间数据的传输通过快速存储节点来完成，以此来降低与后端的通信开销，最后将所有轮的数据合并写入后端的对象存储中。



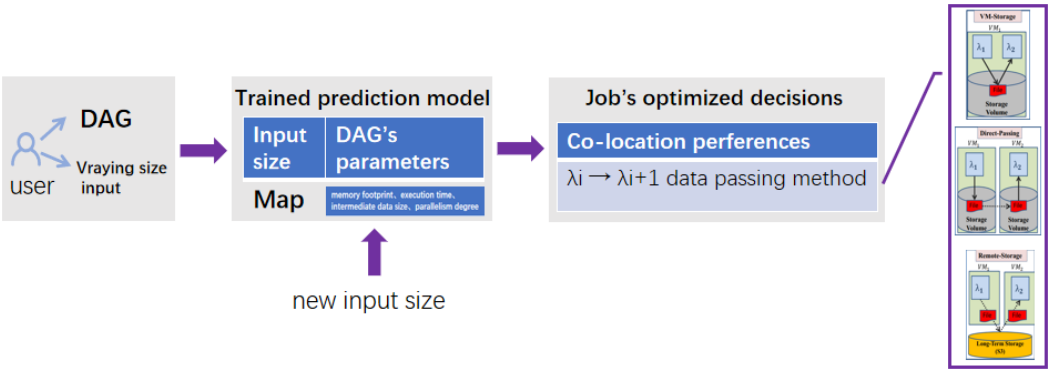
  - 问题：快速存储成本高

Internal storage

- Cloudburst ***[Cloudburst: Stateful Functions-as-a-Service VLDB'20]***
  - 设计：利用各个VM的本地存储作为存储资源实现分布式缓存，为无状态函数提供数据局部性，减少与远端数据库的交互，降低I/O时延



  - 问题：缓存一致性
- SONIC ***[Sonic: Application-aware Data Passing for Chained Serverless Applications ATC'21]***
  - 设计： 根据用户提供的应用DAG和不同大小的输入，训练出输入规模到DAG参数 *(Memory Footprint、Execution Time、Intermediate Data Size、Fanout Degree)* 的映射；当新的输入到达时，根据训练模型确定每个函数的最佳位置以及相邻函数间的最佳传输方式。



- 问题：(自适应的三种数据传输方式)
  - VM-Storage（并发度低时，性能好）
    方案：将发送函数的状态保存在VM的存储中，并将接收函数调度在同一VM上执行
    问题：当接收函数并发度过高时，会导致单个VM负载过重，而且会使接收函数进行排队，计算时间增加。
  - Direct-Passing（没有调度限制，支持更高的并发度）
    方案：将发送函数的输出保存在其VM1存储中，当接收函数被调度在另一个VM2执行时，将数据从VM1拷贝到VM2。
    问题：当不同VM上的接收函数同时获取一台VM上发送函数的输出数据，VM的网络带宽将成为瓶颈。
  - Remote-Storage（没有调度限制，网络带宽大）
    方案：发送函数将输出文件上传到远端存储系统，接受函数执行时下载。
    问题：需要与远端存储系统通信两次，读一次，写一次，通信时延高

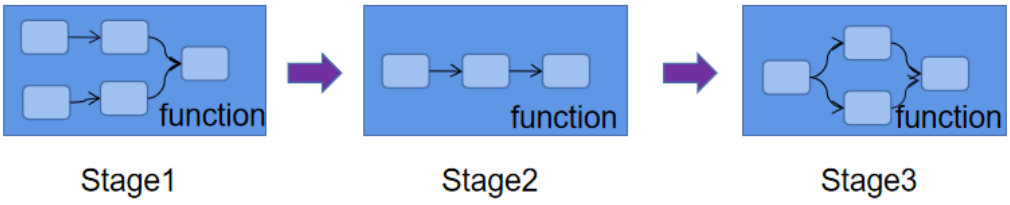
Idea

题目

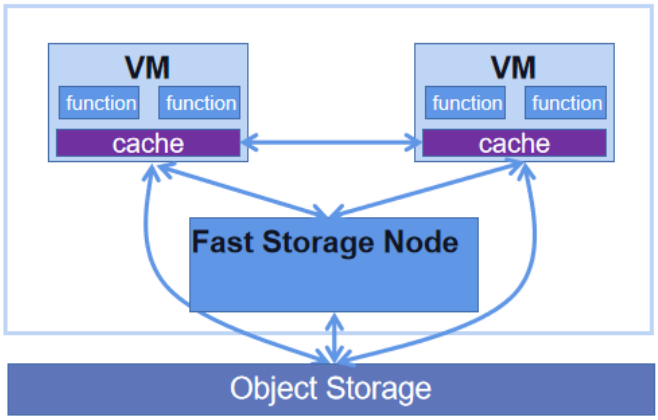
- 通过结合内部存储和外部存储降低Serverless有状态应用的I/O时延

设计

- Data Analytics Application



- Combine Internal storage with External storage
  - 使用Internal cache(VM cache)缓存单个stage中每个function产生的新数据，这样根据动态维护的DAG图就可以从其他VM中通过Direct-Passing的方式直接获取当前task需要的数据，减少与后端数据库的交互
  - 使用External cache(Fast storage node)缓存上个stage产生的中间数据，该stage中的function在该快速存储节点上进行读取和更新，减少与后端数据库的交互
  - 架构图



优势

- **采用集中式缓存储存上一Stage的数据，能否解决数据一致性的问题?**
- 只使用快速存储资源缓存上一Stage的数据，利用各个VM本地存储资源缓存当前Stage内函数产生的新数据，可以降低存储成本
- 直接从其他VM和快速存储节点中获取数据，可以降低I/O时延

挑战

- 如何选择缓存数据？（如何冷热识别（基于DAG图的权重？）？、缓存数据的规模？）
- 跨VM直接传输Direct-Passing，当并发度较高时所导致的网络带宽瓶颈问题。