第六章 新型云计算平台——无服务器计算

2021年9月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

Berkeley View on Cloud Computing in 2009

1. The appearance of infinite computing resources on demand.
2. The elimination of an up-front commitment by cloud users.
3. The ability to pay for use of computing resources on a short-term basis as needed.
4. Economies of scale that significantly reduced cost due to many, very large data centers.
5. Simplifying operation and increasing utilization via resource virtualization.
6. Higher hardware utilization by multiplexing workloads from different organizations.

Eight Issues in Setting up Cloud Environment

1. **Redundancy** for availability, so that a single machine failure doesn't take down the service.
2. **Geographic distribution** of redundant … e the service in case of disaster.
3. **Load balancing** and reque… utilize resources.
4. **Autoscaling** in resp… to scale up or down the system.
5. **Monitoring** t… is still running well.
6. **Logging** to re… needed for debugging or performance tuning.
7. **System upgrades**, …ding security patching.
8. **Migration** to new instances as they become available.

Tedious

What is Serverless Computing?

- Serverless = FaaS + BaaS
  - FaaS: Cloud functions
  - BaaS: services by cloud providers
    - Deployment
    - Fault tolerance
    - Consistency
    - Monitoring
    - …

AWS Lambda

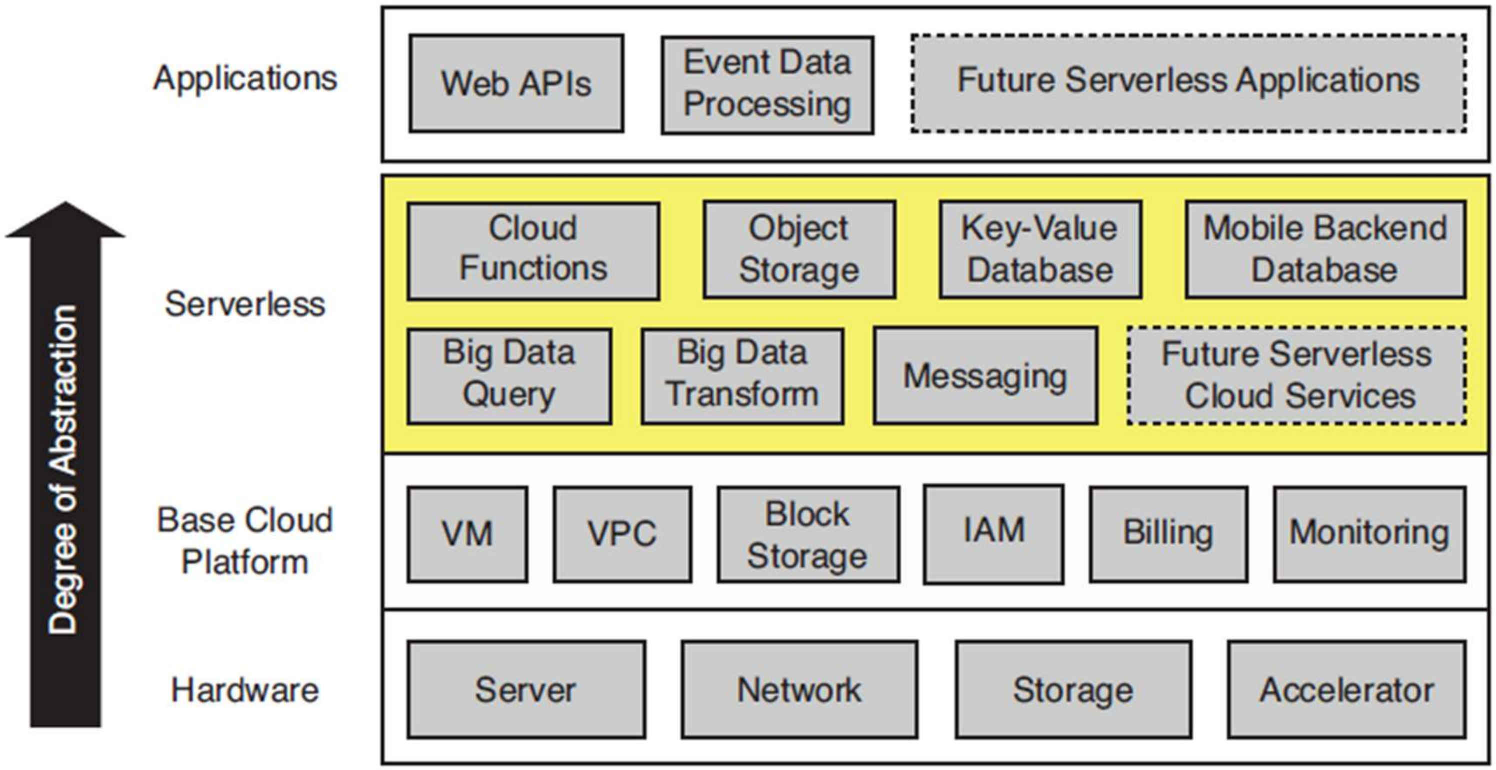
Google Cloud

阿里云

Kubeless

APACHE OpenWhisk™

What is Serverless Computing?

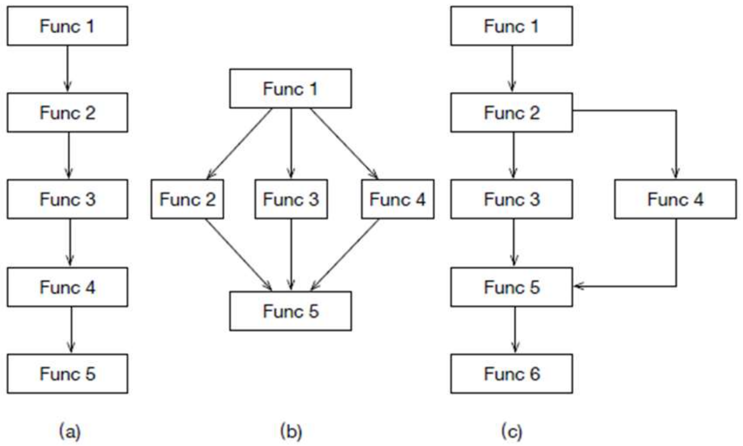


Characteristics

- Function-level management: The basic unit in serverless is *function*.
- Short-running: Functions are expected to complete in a short time period.
- Transparency: Users of serverless are agnostic about the execution environment.
- Stateless: Functions are stateless and only describe the application logic for task processing.
- Pay-as-you-go: the cloud provider charges only when the uploaded functions are actually executed.

Serverless Applications

- Massive and independent parallelism
  - PyWren uses AWS Lambda functions for linear algebra and machine learning hyperparameter optimization.
  - Use AWS Lambda to implement distributed matrix multiplication
  - Serverless version Mapreduce and Spark
- Event-driven handlers
  - The application waits for a specific kind of events.
- General task-based applications



Pros and Cons

- Pros
  - For developers
    - Cost saving.
    - No worrying about deployment and provision.
    - Focus on business logic.
  - For service providers
    - More control over infrastructures.
    - Building a development ecosystem.
- Cons
  - Startup latency.
  - Short-lived execution time.
  - No direct communication.
  - Limited resource, e.g. CPU, memory.
  - No specialized hardware.
  - ...

目录 Contents

SHANGHAI JIAO TONG UNIVERSITY

Case 1: AWS Lambda

- A serverless compute service provided by Amazon since November 2014
- Let cloud users run code without
  - provisioning or managing servers
  - creating workload-aware cluster scaling logic
  - maintaining event integrations
  - managing runtimes
- Upload code as a ZIP file or container image, and run automatically
- Write Lambda functions in most languages (Node.js, Python, Go, Java, and more)
- Trigger from over 200 AWS services and SaaS applications

High-level Architecture

- Frontend
  - Accept users' requests
  - Authentication and authorization
  - Load functions' metadata
- Worker Manager
  - Schedule functions
  - Concurrency control
- Placement
  - Create and maintain function execution slots
- Worker

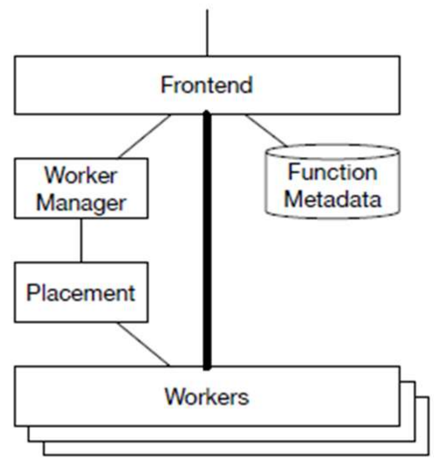


Figure 2: High-level architecture of AWS Lambda event path, showing control path (light lines) and data path (heavy lines)

AWS Lambda Pricing

- The price for Duration depends on the amount of memory you allocate to your function.

	Price
Requests	$0.20 per 1M requests
Duration	$0.0000166667 for every GB-second

- Can allocate any amount of memory to your function between 128MB and 10,240MB, in 1MB increments.

Memory (MB)	Price per 1ms
128	$0.0000000021
512	$0.0000000083
1024	$0.0000000167
1536	$0.0000000250
2048	$0.0000000333

AWS Lambda Pricing

- If you allocated 512MB of memory to your function, executed it 3 million times in one month, and it ran for 1 second each time, your charges would be calculated as follows:
- Monthly compute charges
  - The monthly compute price is $0.00001667 per GB-s and the free tier provides 400,000 GB-s.
  - Total compute (seconds) = 3M * (1s) = 3,000,000 seconds
  - Total compute (GB-s) = 3,000,000 * 512MB/1024 = 1,500,000 GB-s
  - Total compute – Free tier compute = Monthly billable compute GB- s
  - 1,500,000 GB-s – 400,000 free tier GB-s = 1,100,000 GB-s
  - Monthly compute charges = 1,100,000 * $0.00001667 = $18.34

AWS Lambda Pricing

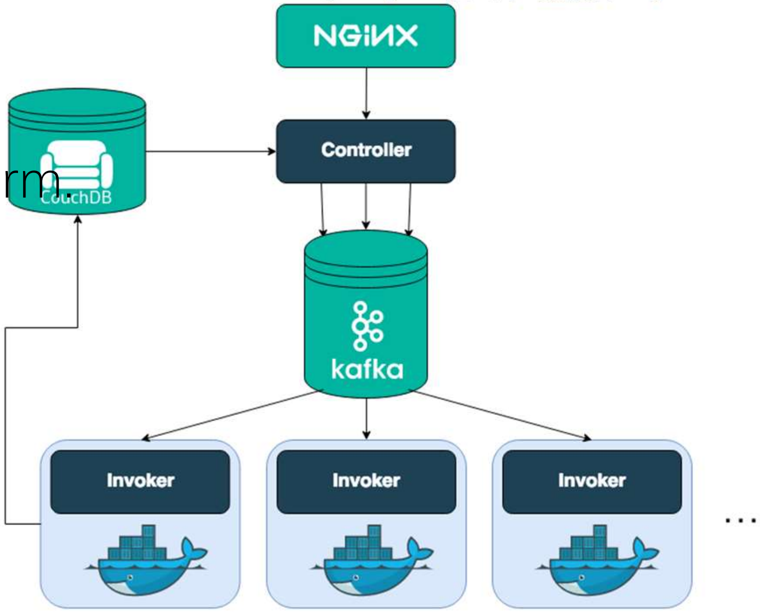
- If you allocated 512MB of memory to your function, executed it 3 million times in one month, and it ran for 1 second each time, your charges would be calculated as follows:
- Monthly request charges
  - The monthly request price is $0.20 per 1 million requests and the free tier provides 1M requests per month.
  - Total requests – Free tier requests = Monthly billable requests
  - 3M requests – 1M free tier requests = 2M Monthly billable requests
  - Monthly request charges = 2M * $0.2/M = $0.40
- Total charges = Compute charges + Request charges = $18.34 + $0.40 = $18.74 per month

Case 2: OpenWhisk

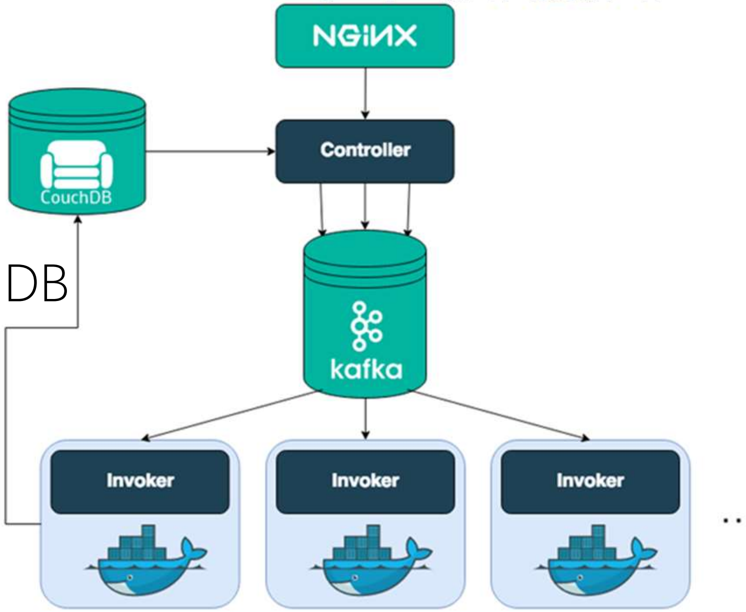
- An open source, distributed Serverless platform
- Manage the infrastructure, servers and scaling using Docker containers
- Characteristics
  - Deploys anywhere
  - Write functions in any language
  - Integrate easily with many popular services
  - Combine your functions into rich compositions
  - Scaling Per-Request & Optimal Utilization

Architecture

- Nginx: a reverse proxy server.
- Kafka: a distributed event streaming platform.
- CouchDB
  - subjects: authentication and authorization.
  - whisks: code, resource requirements.
  - activations: execution results.
- Controller
- Invoker: executing actions.



Procedure of Function Invocation



1. Entering the system: Nginx
2. Entering the system: Controller
3. Authentication and Authorization: CouchDB
4. Getting the action: CouchDB
5. Invoke the action: Controller
6. Forming a line: Kafka
7. Executing the code: Invoker
8. Storing the results: CouchDB

OpenWhisk Demo

1. Create a file named hello.py

```python
def main(dict):
    if 'name' in dict:
        name = dict['name']
    else:
        name = "stranger"
    greeting = "Hello " + name + "!"
    print(greeting)
    return {"greeting": greeting}
```

2. Create an action called helloPy using hello.py

```
$ wsk action create helloPy hello.py
```

```
ok: created action helloPy
```

OpenWhisk Demo

3. Invoke the helloPy action using command-line parameters

```
$ wsk action invoke helloPy --result --param name World
```

```
{
"greeting": "Hello World!"
}
```

4. Additional Resources
   - Using External Python Libraries in OpenWhisk
   - Auto Retweeting Example in Python

目录 Contents

Limitations of Serverless Computing Platforms

1. Inadequate storage for fine-grained operations
2. Lack of fine-grained coordination
3. Poor performance for standard communication patterns
4. Predictable Performance

Limitation 1: Storage

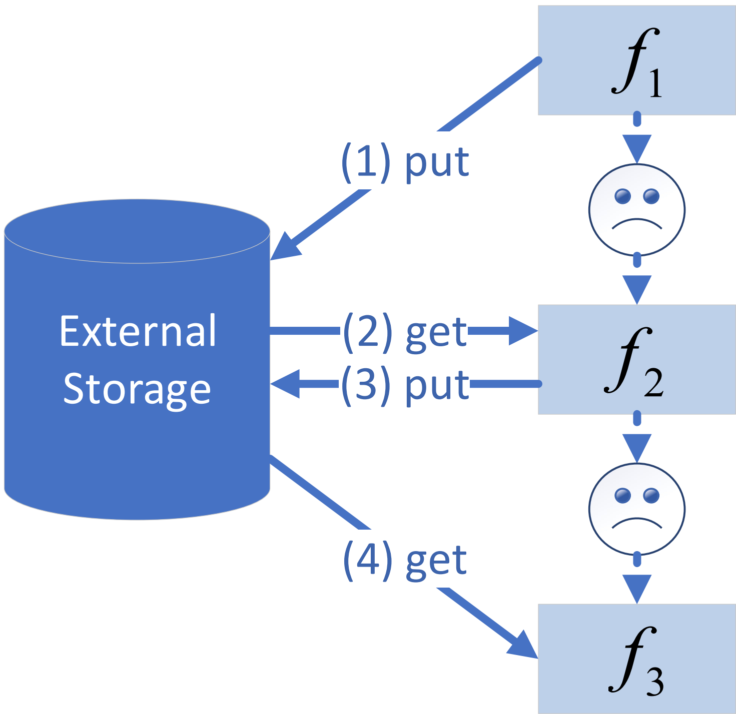
- Difficult to support applications that have fine-grained state sharing needs
- Object storage services
  - Including AWS S3, Azure Blob Storage, and Google Cloud Storage
  - Highly scalable and provide inexpensive long-term object storage
  - High access costs and high access latencies
- Key-value databases
  - Such as AWS DynamoDB, Google Cloud Datastore
  - Provide high IO Per Second (IOPS)
  - Expensive and can take a long time to scale up
  - Not fault tolerant and not autoscale

Limitation 1: Storage

		Block Storage (e.g., AWS EBS, IBM Block Storage)	Object Storage (e.g., AWS S3, Azure Blob Store, Google Cloud Storage)	File System (e.g., AWS EFS, Google Filestore)	Elastic Database (e.g., Google Cloud Datastore, Azure Cosmos DB)	Memory Store (e.g., AWS Elas-tiCache, Google Cloud Memorys-tore)	**"Ideal" storage service for serverless computing**
Cloud functions access		No	Yes	Yes[13]	Yes	Yes	Yes
Transparent Provisioning		No	Yes	Capacity only[14]	Yes[15]	No	Yes
Availability and persistence guarantees		Local & Persistent	Distributed & Persistent	Distributed & Persistent	Distributed & Persistent	Local & Ephemeral	Various
Latency (mean)		< 1ms	$10 - 20$ms	$4 - 10$ms	$8 - 15$ms	< 1ms	< 1ms
Cost[16]	Storage capacity (1 GB/month)	\$0.10	\$0.023	\$0.30	\$0.18–\$0.25	\$1.87	~\$0.10
Cost[16]	Throughput (1 MB/s for 1 month)	\$0.03	\$0.0071	\$6.00	\$3.15-\$255.1	\$0.96	~\$0.03
Cost[16]	IOPS (1/s for 1 month)	\$0.03	\$7.1	\$0.23	\$1-\$3.15	\$0.037	~\$0.03

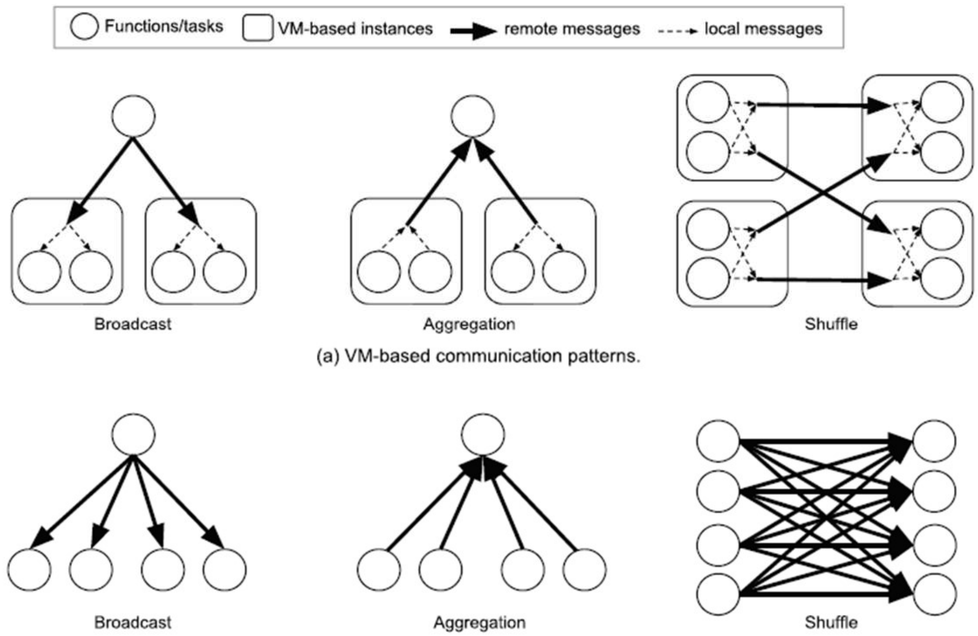
Limitation 2: Coordination

- Requirement: If task A uses task B's output, there must be a way for A to know when its input is available.
- None of the existing cloud storage services come with notification capabilities.
- Current methods
  - manage a VM-based system that provides notifications
  - implement their own notification mechanism



Limitation 3: Communication

- Broadcast, aggregation, and shuffle are some of the most common communication primitives in distributed systems.
- Communication patterns for these primitives for both VM-based and function-based solutions.

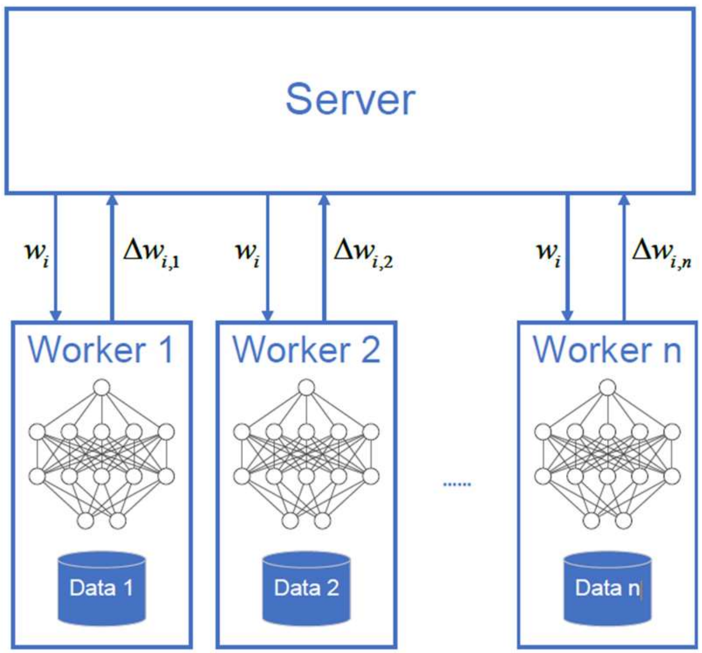


(a) VM-based communication patterns.

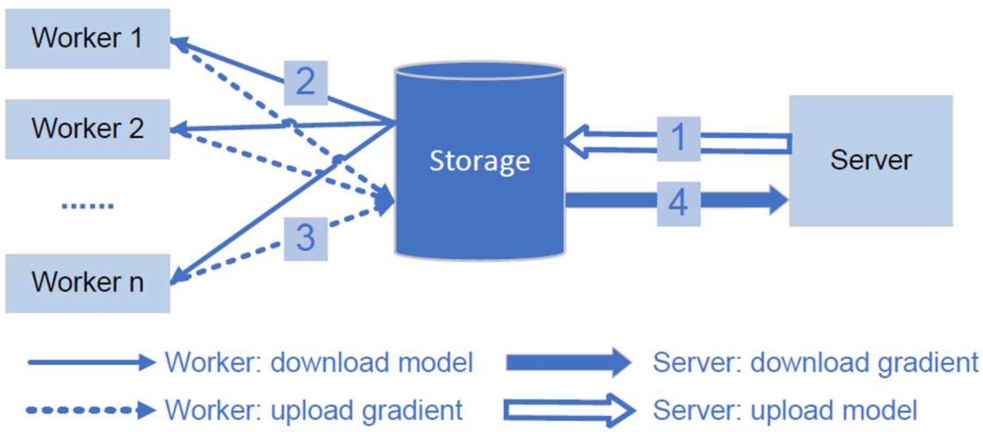
(b) Function-based communication patterns.

Case: Distributed Machine Learning

- Parameter Server



- Serverless Parameter Server



Feasible Optimization for Communication

- Optimizing the storage server
  - Current storage services designed for short-running functions and thus become a performance bottleneck.
  - Pocket introduces multi-tier storage including DRAM, SSD and HDD.
  - Locus also combines different kinds of storage devices to achieve both performance and cost-efficiency for serverless analytics
- Optimizing the communication path
  - Optimize the communication path when the relationship between functions is known in advance.
  - Another line of work tries to kick the storage server out of the communication path with network mechanisms.

Limitation 4: Cold Start

- Cold start latency
  - the time it takes to start a cloud function
  - the time it takes to initialize the software environment
  - application-specific initialization in user code
- Feasible optimization for cold start
  - Container cache: When a function is finished, the serverless framework can retain its runtime environment.
  - Pre-warming: OpenWhisk can pre-launch Node.js containers if it has observed that the workload mainly consists of Node.js-based functions.
  - Container optimization: Provide lean containers with much faster boot time than vanilla ones
  - Looking for other abstractions: Google gVisor, AWS FireCracker, Unikernel

目录 Contents

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Related Works

- Optimizing the storage server
  - Pocket
  - **Locus**
- Optimizing the communication path
  - SAND
- Serverless ML Training
  - Siren
  - Cirrus
- Serverless ML Inference
  - Gillis

Optimizing the storage server

- Pocket: Elastic Ephemeral Storage for Serverless Analytics

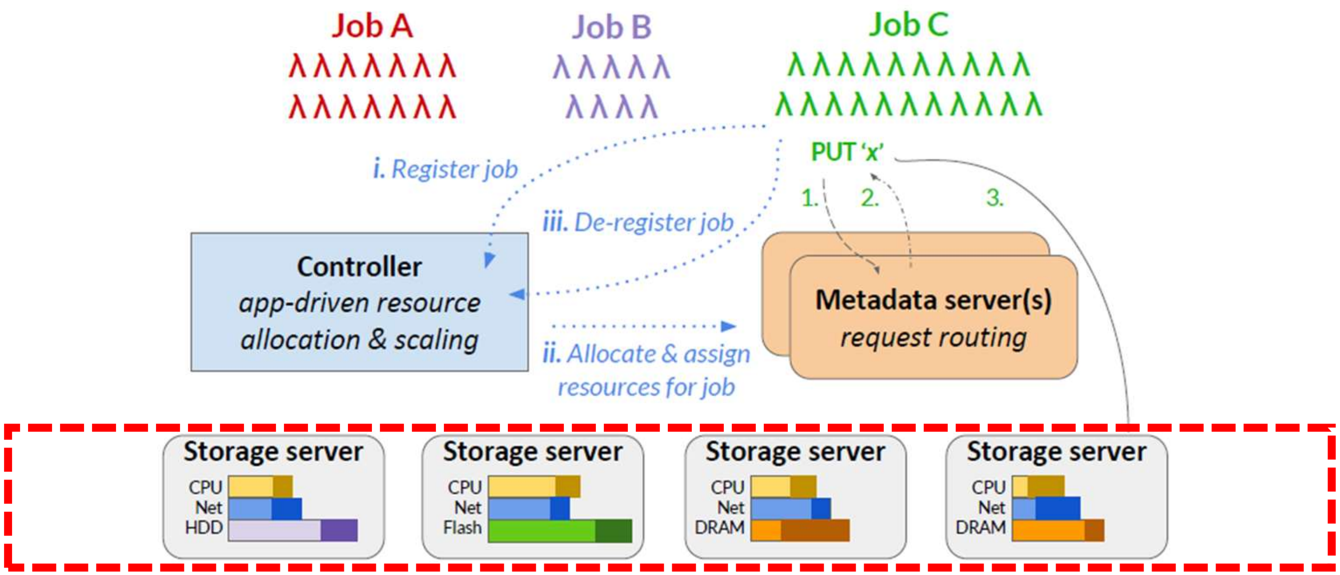
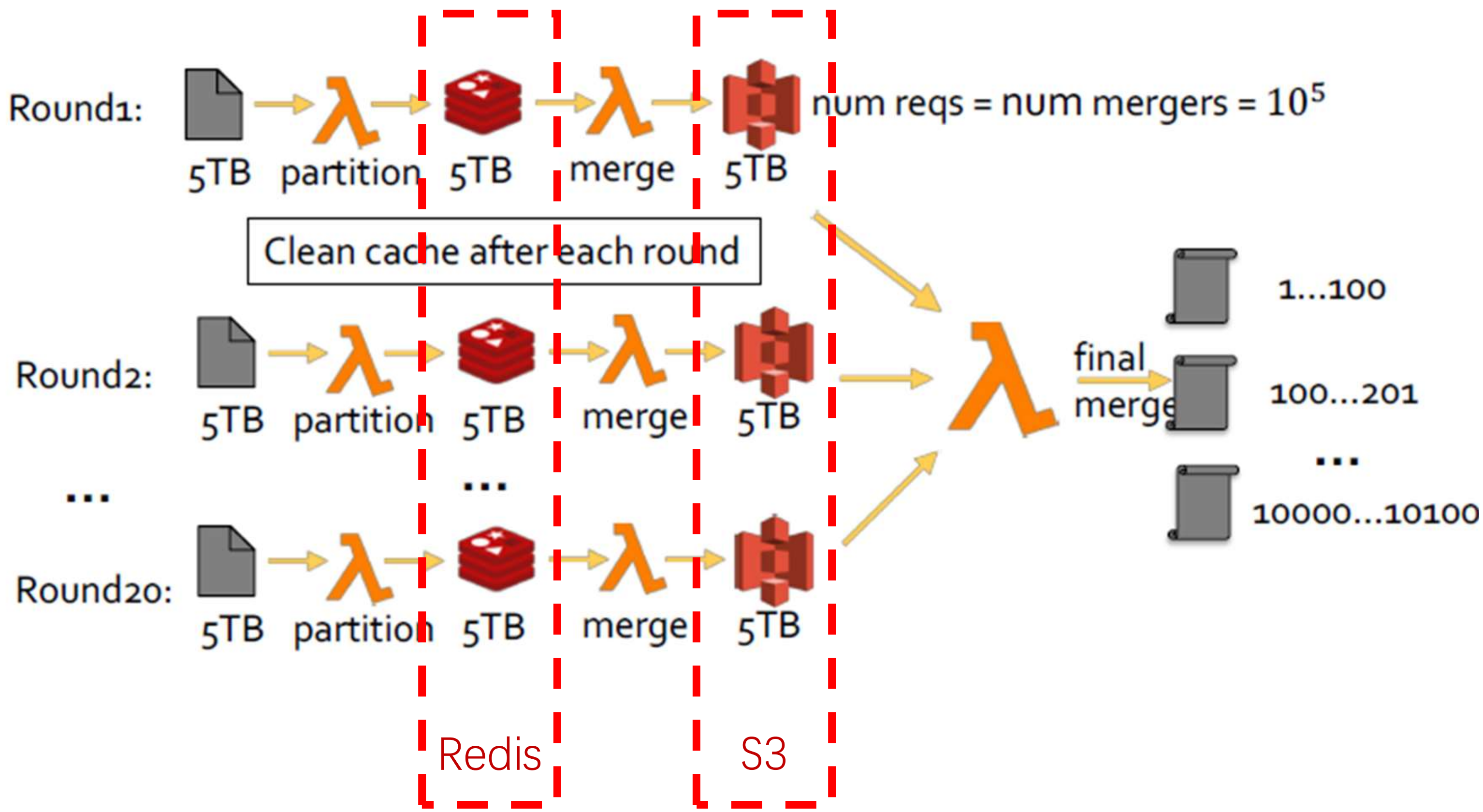


Figure 4: Pocket system architecture and the steps to register job C, issue a PUT from a lambda and de-register the job. The colored bars on storage servers show used and allocated resources for all jobs in the cluster.

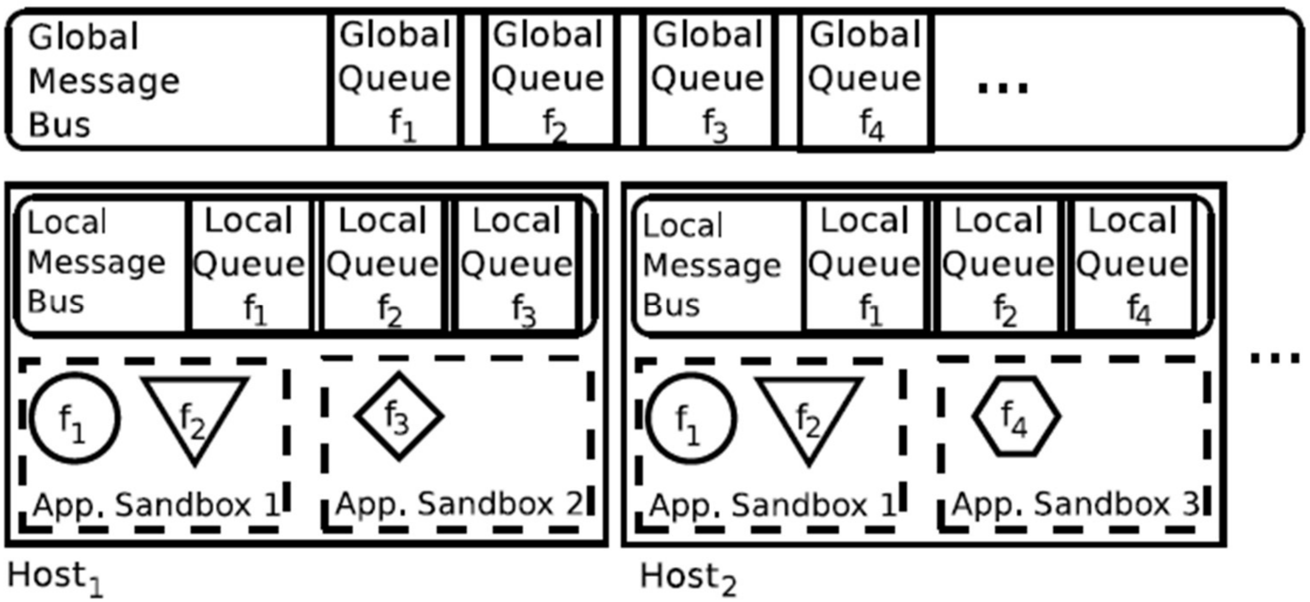
Optimizing the storage server

- Locus: Shuffling Fast and Slow on Serverless Architecture



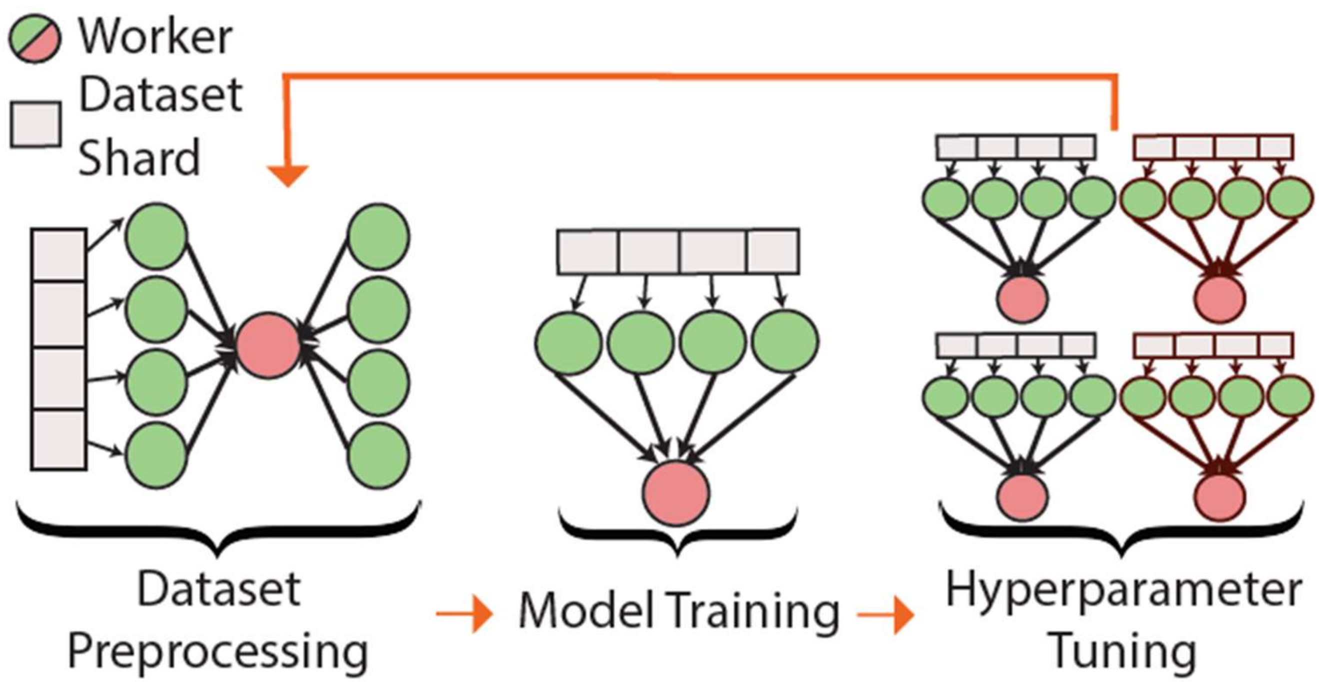
Optimizing the communication path

- SAND: Towards High-Performance Serverless Computing



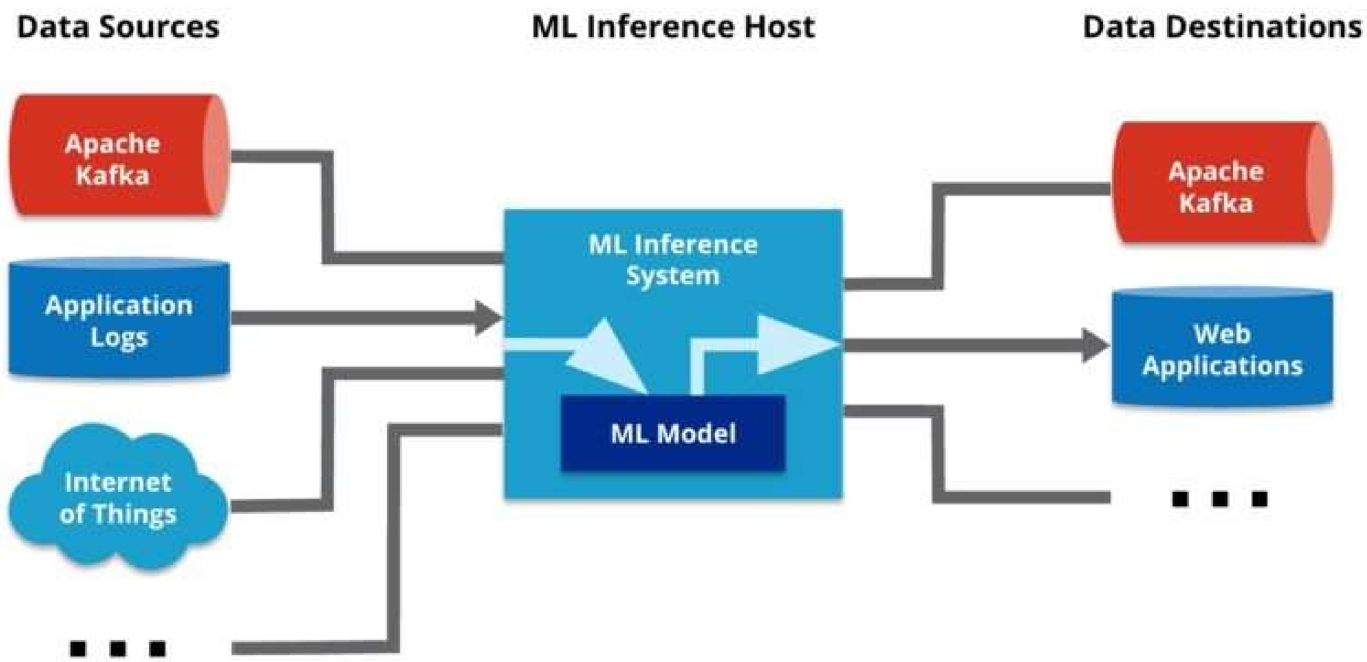
Serverless Computing and Machine Learning

- Training Stage



Serverless Computing and Machine Learning

- Inference Stage



Serverless Computing and Machine Learning

- Training Stage
  - Computing intensive
  - Parallel execution
- Inference Stage
  - High throughput
  - Low latency
  - High availability
  - Other SLA requirements

→ Serverless Computing

Siren

Distributed Machine Learning with a Serverless Architecture

Hao Wang[1], Di Niu[2] and Baochun Li[1]
[1]*University of Toronto*, {haowang, bli}@ece.utoronto.ca [2]*University of Alberta*, dniu@ualberta.ca

Siren

- Motivation
  - parallel computing
  - variant resource requirement
  - trial-and-error
- Contribution
  - combine serverless computing and machine learning
  - utilize reinforcement learning for resource scheduling
  - reduce job completion time by 44% for training jobs

Siren

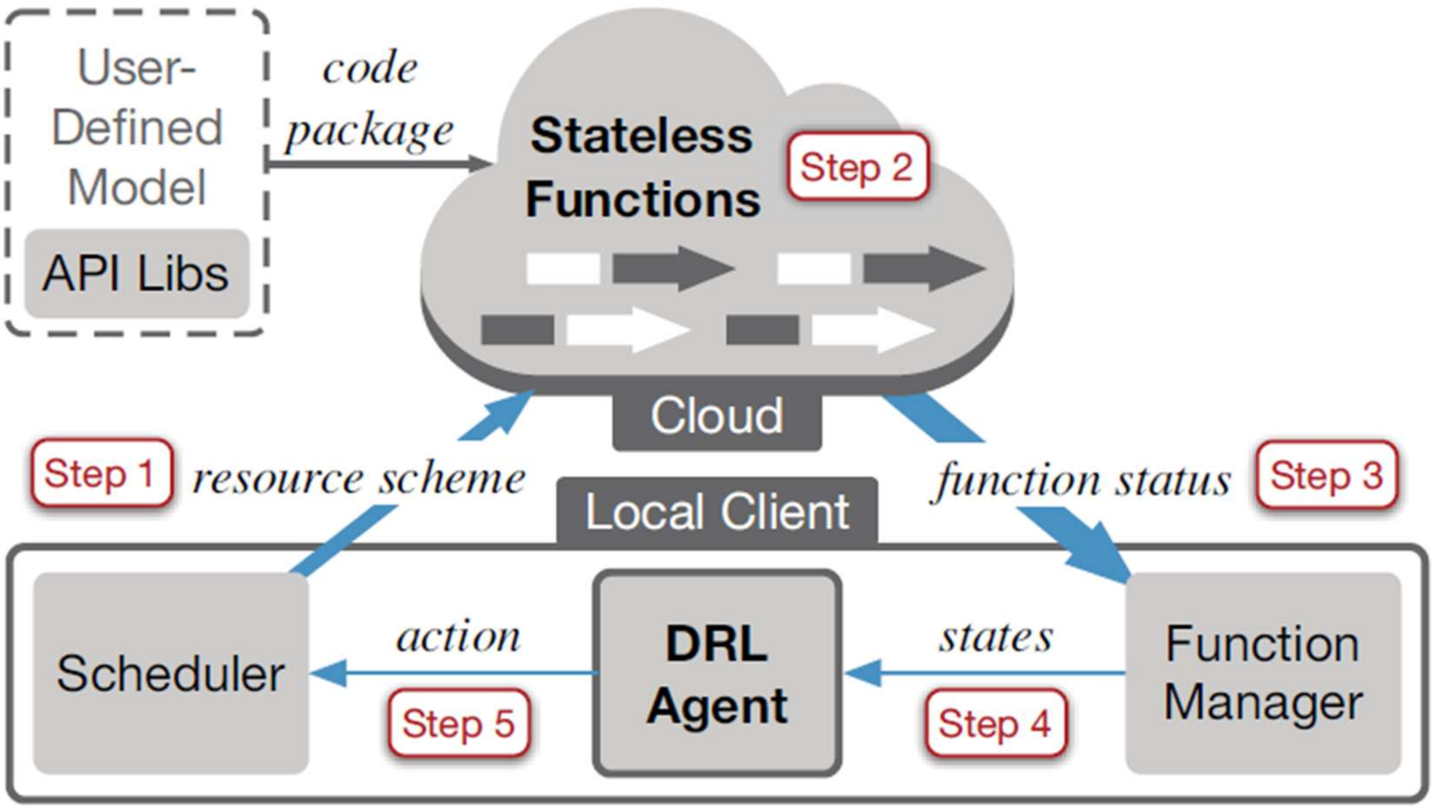


Fig. 2: The system architecture and workflow of SIREN.

Cirrus

CIRRUS: a Serverless Framework for End-to-end ML Workflows

Joao Carreira
University of California, Berkeley
joao@berkeley.edu

Pedro Fonseca
Purdue University
pfonseca@purdue.edu

Alexey Tumanov
Georgia Institute of Technology
atumanov@gatech.edu

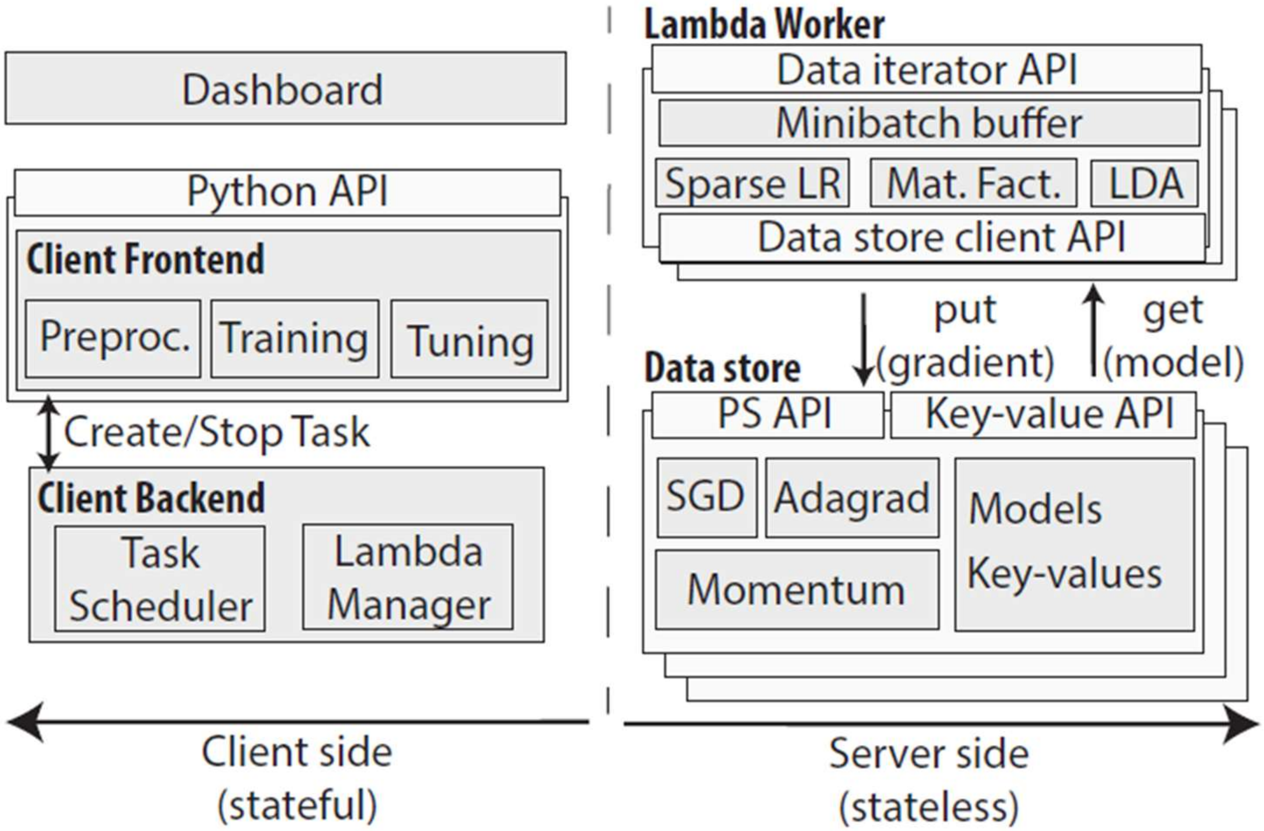
Andrew Zhang
University of California, Berkeley
andrewmzhang@berkeley.edu

Randy Katz
University of California, Berkeley
randykatz@berkeley.edu

Cirrus

- Machine Learning
  - Over-provisioning
  - Explicit resource management
- Serverless Computing
  - Small local memory and storage
  - Low bandwidth and lack of P2P communication
  - Short-lived and unpredictable launch times
  - Lack of fast shared storage

Cirrus


Dashboard
Python API
Client Frontend
Preproc.
Training
Tuning
Create/Stop Task
Client Backend
Task Scheduler
Lambda Manager
Lambda Worker
Data iterator API
Minibatch buffer
Sparse LR
Mat. Fact.
LDA
Data store client API
put (gradient)
get (model)
Data store
PS API
Key-value API
SGD
Adagrad
Momentum
Models
Key-values
Client side (stateful)
Server side (stateless)

Gillis

- *Best Paper Runner Up* of IEEE ICDCS 2021

Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning

Minchen Yu*, Zhifeng Jiang*, Hok Chun Ng*, Wei Wang*, Ruichuan Chen†, Bo Li*
*Hong Kong University of Science and Technology
{myuaj, zjiangaj, hcngac, weiwa, bli}@cse.ust.hk
†Nokia Bell Labs
ruichuan.chen@nokia-bell-labs.com

Gillis

- Problem: Serverless functions have constrained resources in CPU and memory, making them inefficien or infeasible to serve large neural networks.
- Design
  - Fork-join computing model
  - Coarse-grained model grouping
  - Two model partition algorithms

Fig. 4: The fork-join model for function coordination.

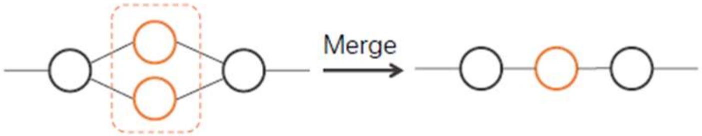
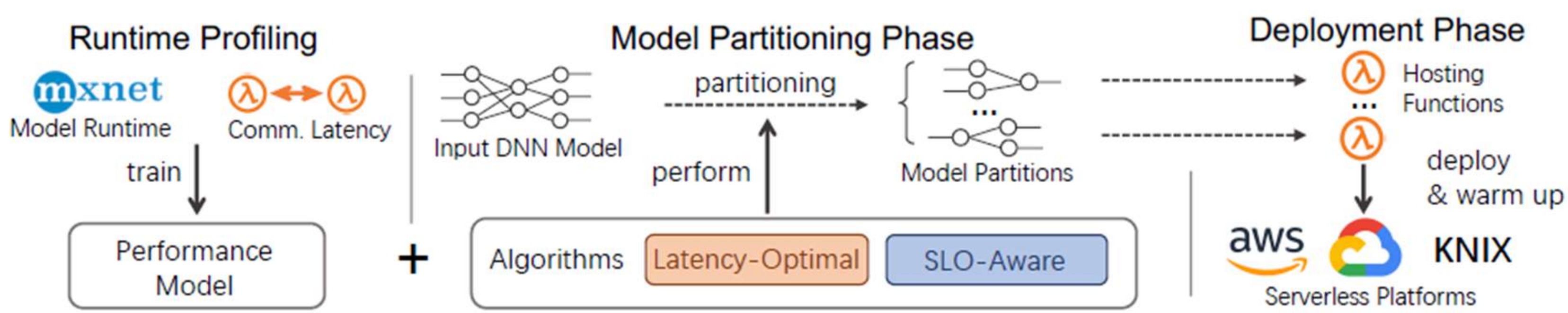


Fig. 5: An illustration of branch merging, where two parallel branch modules are merged into one layer.

Gillis

- Workflow
  - Runtime Profiling
  - Model Partition
    - Latency-optimal algorithm
    - SLO-aware algorithm
  - Deployment



谢谢！