Caerus: NIMBLE Task Scheduling for Serverless Analytics

Hong Zhang, Yupeng Tang, Anurag Khandelwal, Jingrong Chen, Ion Stoica
Serverless computing

AWS Lambda  Google Cloud Functions  Azure Functions  IBM Cloud Functions

Fast Scaling

30 ~ 120 Seconds  < 1 Second

Fine-grained billing

Per second  Per millisecond
## Serverless analytics

<table>
<thead>
<tr>
<th>Web severing</th>
<th>IoT applications</th>
<th>PyWren (SoCC'17)</th>
<th>gg compiler (ATC'19)</th>
<th>Locus (NSDI'19)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video processing</td>
<td>Scaling python functions</td>
<td>Distributed compiling</td>
<td>Mapreduce &amp; SQL-like queries</td>
<td></td>
</tr>
</tbody>
</table>

2014 Single-stage functions | 2017 General data analytics | 2021

### Job Execution plan

- **Job completion time (JCT)**
- **Deploy**
- **Cost (total duration of all workers)**

A pool of (infinite) serverless workers
Serverless scheduling: a new problem

Inter-job scheduling
Optimization Metrics: average JCT, cluster utilization, fairness across jobs

Now handled by the serverless platform

Intra-job scheduling across tasks
Optimization Metrics: Both JCT and cost for each individual job

Can existing server-centric intra-job scheduling policies optimize both JCT and cost in serverless settings?
Trade-off: Lazy vs. Eager

Lazy: start a task after ALL tasks in its upstream stages have finished (Spark)

A MapReduce job with 3 map tasks and 3 reduce tasks

Job completion time: $16 + 15 = 31$

Cost (total duration): $(12 \times 2 + 16) + (2 + 7 + 15) = 64$
Trade-off: Lazy vs. Eager

**Lazy**: start a task after **ALL** tasks in its upstream stages have finished (Spark)

- **Map tasks**: 
  - 12
  - 16
  - 12

- **Reduce tasks**: 
  - 2
  - 7
  - 15

Stage barrier

Job completion time: 16 + 15 = 31
Cost (total duration): (12*2+16)+(2+7+15) = 64

- Minimizes cost (duration)
- Much longer job completion time (1.63X)

**Eager**: start a task when **ANY** output from its upstream stages is ready (Mapreduce Online)

- **Map tasks**: 
  - 12
  - 16
  - 12

- **Reduce tasks**: 
  - 1
  - 2
  - 3

The part (e.g., data aggregation) which can only start after receiving all the mapper output.

Job completion time: 16 + 3 = 19
Cost: (12*2+16) + 16*3 + (1+2+3) = 94

- Minimizes job completion time
- Much higher cost (1.47X)
NIMBLE scheduling: main idea

Main idea:
- Fully exploit the **flexible resource scaling** of serverless computing
- Calculate and enforce the **best launch time** for each individual task

How to calculate the optimal launch time for each task?

Job completion time: $16 + 3 = 19$
Cost (total duration): $(12 \times 2 + 16) + (1 + 5 + 12) + (1 + 2 + 3) = 64$

Lazy
- $(64, 31)$

Eager
- $(94, 19)$

NIMBLE
- $(64, 19)$
Challenge 1: Describe pipelinability

- NIMBLE scheduling requires a precise description of the pipelinability across different job stages.

Map tasks  Reduce tasks

Stage-level DAG:

Cannot calculate the optimal task launch time without sub-stage level information.

How to describe pipelinability at sub-stage level?
Challenge 2: Arbitrary DAGs

- General analytics workloads can have complicated DAGs.
  - **Within a stage:** tasks can consume data from multiple upstream stages
  - **Across stages:** tasks can have cascading dependencies

---

How to calculate the optimal task launch time for arbitrary DAGs?
**NIMBLE design outline**

- **Challenge 1:** How to describe pipelinability at sub-stage level?
  - Develop a *step model* to precisely capture the sub-stage level pipelinability

- **Challenge 2:** How to calculate the optimal task launch time for arbitrary DAGs?
  - Develop a scheduling algorithm which guarantees *optimal cost* while being *Pareto-optimal* between cost and JCT for arbitrary DAGs
**Step model**

- **Idea:** Break stages into steps
  - Step: largest pipeline-able component within a stage
  - Separated by pipeline breakers\(^1\) (e.g., MIN, MAX, SUM)

1. A pipeline breaker is an operator that produces the first output only after all its input has been processed.
**Step model**

- **Idea:** Break stages into steps
  - Step: largest pipeline-able component within a stage
  - Separated by pipeline breakers\(^1\) (e.g., MIN, MAX, SUM)

![Diagram showing the step model with map and reduce tasks, pipeline breakers, and data dependencies within and across stages.](image)

1. A pipeline breaker is an operator that produces the first output only after all its input has been processed.
Step model

• Example: the step model for a complicated SQL query in TPC-DS benchmark

---

Step model can efficiently describe pipelinability across a wide range of applications
Basic algorithm for 2-stage map-reduce

• Intuition to calculate the launch time:
  • Optimally overlap the parent-child step pair based on the data produce and data consume rate

![Diagram showing map and reduce tasks with overlap]

Historical + online job information
Basic algorithm for 2-stage map-reduce

- Optimal launch time in three simple steps
  - Step 1: Calculate *optimal task duration* based on *Lazy* solution

![Diagram showing map and reduce tasks with optimal duration calculation]
Basic algorithm for 2-stage map-reduce

- Optimal launch time in three simple steps
  - Step 2: Calculate \textit{optimal task finish time} based on \textit{Eager} solution
• **Optimal launch time in three simple steps**
  • **Step 3:** Calculate the task launch time $t^*$ as:
    
    $$\text{optimal task finish time} - \text{optimal task duration}$$

---

Basic algorithm for 2-stage map-reduce

- **Map tasks**
- **Reduce tasks**
  - Can be pipelined with map
  - Cannot be pipelined with map

```
λ 12
λ 16
λ 12
```

```
1 5 12
1 2 3
```

- Optimal finish time
- Optimal duration
Basic algorithm for 2-stage map-reduce

- Optimal launch time in three simple steps

Theorem 1: \( t^* \) ensures optimal cost and finish time for each reduce task.
From map-reduce to arbitrary DAGs

• **Challenges for arbitrary DAGs:**
  - Within a stage: tasks can consume data from multiple upstream stages
  - Across stages: tasks can have cascading dependencies

• **Takeaways:**
  - **Bad news:** *Impossible* to design an algorithm that can achieve optimal cost and JCT *simultaneously* for arbitrary DAGs
  - **Good news:** Extend the basic algorithm to guarantee *optimal cost* while being *Pareto-optimal* between cost and JCT
Caerus System

• Caerus: a task-level scheduler for serverless analytics which enables NIMBLE scheduling

Data Analytics Framework with Caerus
Evaluation results on AWS

TPC-DS (4 queries)

BigData (3 queries)

NIMBLE scheduling can effectively optimize both JCT and cost across all these workloads.
Takeaways

- Serverless analytics introduces a new intra-job scheduling problem to optimize both JCT and cost
  - Existing solutions expose a hard tradeoff between these two metrics

- NIMBLE scheduling with a simple idea: to launch each task at its right time
  - Step model to capture sub-stage level pipelinablility and data dependencies
  - Achieves cost optimality while being Pareto-optimal between cost and JCT

- Caerus: a task-level scheduler for serverless analytics which enables NIMBLE scheduling in practice

Thank you! Contact email: hongzhangblaze@gmail.com