GeoClone: Online Task Replication and Scheduling for Geo-Distributed Analytics under Uncertainties

Tiantian Wang[1], Zhuzhong Qian[1], Lei Jiao[2], Xin Li[3], Sanglu Lu[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, China
[2]Department of Computer and Information Science, University of Oregon, USA
[3]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China

***Abstract*—The execution and completion of analytics jobs can be significantly inflated by the slowest tasks contained. Despite task replication is well adopted to reduce such straggler latency, existing replication strategies are unsuitable for geo-distributed analytics environments that are highly dynamic, uncertain, and heterogeneous. In this paper, we firstly model the task replication and scheduling problem over time, capturing the geo-analytics features. Afterwards, we design an online algorithm, GeoClone, to select tasks to replicate and select sites to execute the task replicas in an irrevocably online manner, through jointly considering the execution progress of each job and the resource performance in each site. We rigorously prove the competitive ratio to exhibit the theoretical performance guarantee of GeoClone, compared against the offline optimal algorithm which knows all the inputs at once beforehand. Finally, we implement GeoClone with Spark and Yarn for experiments and also conduct extensive large-scale simulations, which confirms GeoClone's practical superiority over multiple state-of-the-art replication strategies.**

I. Introduction

While data analytics is essential for enterprises and service providers, the execution and completion of the analytics jobs, which often consist of multiple tasks, can be dragged by the slowest tasks. An analytics job only completes when all of its tasks finish execution, and a task can only start to execute when all the tasks that it depends on have finished execution beforehand. In fact, the straggler tasks can inflate the execution time of analytics jobs by 34% at median, as reported for a Microsoft production cluster. The causes of the stragglers are complex and inevitable [1]–[3], including component failures, resource contention, and network congestion. As analytics systems are extending to a much larger *geo-distributed* scale, the aforementioned factors can only escalate fiercely.

One widely adopted approach to taming stragglers for low-latency analytics is *task replication* [1]–[7], which executes multiple replicas for a task and, as one of these replicas completes, kills the other replicas immediately. As a task is replicated, an execution race occurs between the original and the replicated tasks, and the completion of the task relies on its fastest replica. For geo-distributed analytics that involve edge servers, one may desire to replicate tasks not just locally, but also remotely, because edge servers often have limited resources, intermittent connectivity (e.g., for servers colocated with cellular base stations), and imbalanced workloads [8]–[10], which all impact task executions. Fig. 1 exhibits a job

978-1-7281-6887-6/20/$31.00 ©2020 IEEE

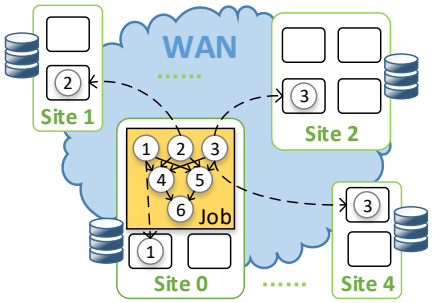


Fig. 1: Geo-distributed analytics job execution

consisting of 6 tasks represented as a Directed Acyclic Graph (DAG). Based on the decisions of the scheduler, some tasks are replicated and dispatched for execution across the 4 sites.

Unfortunately, replicating and scheduling tasks on the fly across sites actually faces critical challenges. First, the system environments are often highly dynamic and uncertain. For instance, the analytics job arrivals can fluctuate and be unpredictable; the resource availabilities can also keep changing due to the occupation by other (non-analytics) jobs [11]–[13]. It is therefore non-trivial to devise decision-making algorithms that can constantly adapt to such dynamics and uncertainties. Second, the system environments are also vastly heterogeneous with unknown resource performance [14]–[17]. For example, platform heterogeneities and data (i.e., inputs for the tasks) locations can make the execution time of a replica site-dependent; even in the same site, the execution time can vary as time goes and be unknown before a replica is actually executed. These factors increase the difficulty for making appropriate scheduling decisions. Third, any task replication and scheduling mechanism should not impose excessive overhead (e.g., the number of replicas used) itself to the system. It is important to strike the balance between the complexity of the algorithm/mechanism and the speedup of the jobs/tasks.

Existing task replication and scheduling strategies are inapplicable and also insufficient in face of the aforementioned challenges. *Detection-based* strategies [1], [2], [4], [5] spend considerable time and efforts identifying the stragglers, and such actions and overhead largely limit the efficacy in using replicas to speed up jobs. *Clone-based* strategies [3] replicate tasks as they are launched and reserve extra resources, which impairs the resource availability for other jobs and the system efficiency; further, they [6], [7], [18] often consider homoge-

neous, single-site environments only, which is incompatible with the heterogeneous, distributed geo-analytics systems. Overall, existing works also have not performed any comprehensive study of this problem regarding handling system uncertainties from both theoretical and practical perspectives.

In this paper, firstly, we model and formulate the task replication and scheduling problem in the geo-analytics systems, capturing the arbitrary system dynamics including job arrivals, task inter-dependencies, and resource availabilities. We introduce randomness to represent resource performance heterogeneities and variations in order to align with real-world production system statistics [19]. Our models feature the fact that the completion of a task is determined by its fastest replica and the completion of a job is determined by its slowest task. We exhibit that our problem is NP-hard, even in the offline setting where all dynamic inputs are given at once beforehand.

Then, we design and present a polynomial-time, online task replication and scheduling algorithm, *GeoClone*. GeoClone responds to unpredictable system uncertainties by periodically and selectively replicating and executing a subset of current tasks in selected sites. GeoClone consists of two steps. First, it estimates an upper bound for the number of replicas to be spawned for the selected tasks (without violating task dependencies) by trying to use out the current available computing slots, based on the principles of the Shortest Remaining Processing Time and the fair slot sharing [20], [21], while considering the system utilization. Second, GeoClone selects the current best sites (without violating per-site resource availability) for those tasks to run the replicas while reducing the estimated upper bound for the replica numbers to a carefully-set threshold and prioritizing the tasks that get closer to the job completion. Overall, GeoClone works for uncertain and heterogenous environments without detecting the stragglers.

Further, we rigorously prove the theoretical performance guarantee of GeoClone. While it has been found that no online algorithm can achieve a finite *competitive ratio* for a class of problems which includes our task replication and scheduling problem, we do not have to be over pessimistic—based on our insight that the "marginal benefit" [21], [22] (i.e., the improvement of the task speed) of spawning new replicas decreases as the number of existing replicas becomes larger, we derive that GeoClone can actually achieve the average response time of all the jobs no longer than $O(\frac{1}{\varepsilon})$ times that achieved by the offline optimal algorithm which is assumed to know all system dynamics beforehand, if the resources (used for executing the jobs/tasks) considered by GeoClone are $1+\varepsilon$ times *faster* than those considered by the offline optimum [23], [24], for any $0<\varepsilon<1$. This result, in turn, guides our design of GeoClone with the ε-based threshold for task replications.

Finally, we conduct extensive experiments via both *implementation* and *simulation* to validate the practical performance of GeoClone. We implement GeoClone as a middleware with the real-world analytics systems Spark [25] and Yarn [26], and simulate large-scale evaluations using the CloudSim [27] simulator. With both real-world [2], [3], [5], [26] and synthetic data traces under a variety of settings, our results indicate that GeoClone consistently improves the average job response time by $40\%\sim65\%$ compared to multiple existing state-of-the-art replication strategies [1]–[3], [8], [16]. Besides, we show that GeoClone (1) achieves performance improvements regardless of systems' utilizations, (2) consumes a very moderate number of replicas, (3) performs well as system heterogeneity and scale increase, and (4) incurs only negligible other overhead.

II. Model and Problem Formulation

Cloud-Edge System, Jobs, and Tasks: We consider a cloud-edge system of multiple geo-distributed "sites", denoted as $\mathcal{K}$, which include both cloud datacenters and edge clusters and connect to one another via WAN. We consider a set of "jobs" $\mathcal{J}$ arriving at the system sequentially, where each job $j \in \mathcal{J}$ consists of a set of "tasks" $\mathcal{L}_j$. Let M_k be the number of "computing slots" (or simply "slots") in site $k \in \mathcal{K}$, where each slot can host one task for execution. We study the system over a time horizon $\mathcal{T}$ of a series of consecutive "time slots". We denote that job $j, \forall j$ arrives at time slot $a_j \in \mathcal{T}$.

Task Dependency: Tasks that belong to the same job may depend on one another. For two tasks l_1 and l_2 of a job, if l_2 depends on l_1, then we write $l_1 \leqslant l_2$. A task cannot start to execute until all the tasks that it depends on complete. Thus, for task l' of job j, if we use $e^j_{l'}$ to denote its execution time (i.e., the amount of time it takes to complete) and $y^j_{l'}$ to denote its starting time slot, respectively, then we have

$$y^j_l \geq y^j_{l'} + e^j_{l'}, \forall l' \leqslant l, \forall l, l' \in \mathcal{L}_j, \forall j, \tag{1}$$

for any task l of job j.

Task Replication: We maintain multiple replicas for each task. We use $x^j_{l,k} \in \mathbb{N}$ to denote the number of replicas of task l of job j placed and executed at site k. Accordingly, we write $\bar{\boldsymbol{x}}^j_l = \{x^j_{l,k} | k \in \mathcal{K}\}$ to represent the numbers of the replicas of task l of job j over all the sites. We have

$$x^j_l = \textstyle\sum_{k\in\mathcal{K}} x^j_{l,k} \geq 1, \forall l \in \mathcal{L}_j, \forall j. \tag{2}$$

Replica Execution Time: We use p^j_l to denote the "intrinsic" execution time of task l of job j when the data it needs are cached locally. As a replica of task l of job j runs in site k, the actual execution execution time $e^j_{l,k}$ can be

$$e^j_{l,k} = p^j_l \cdot s_k + d^j_{l,k}, \tag{3}$$

where $d^j_{l,k}$ is the time to fetch the needed data, possibly from other sites, and s_k is a random scaling factor that stretches the intrinsic execution time, due to the hardware/software heterogeneity at site k and other factors. Table 2 is the scaling factor's distribution of a Facebook's Hadoop cluster [19].

s_k	1	2	3	4	5	6	7	8	9	...
Prob.(%)	23	14	9	3	8	10	4	14	12	...

Fig. 2: The scaling factor's distribution [19].

Task and Job Response Time: We define the (expected) task response time of task l of job j as

$$e^j_l(\bar{\boldsymbol{x}}^j_l) = \mathbb{E}\left[\min_{k\in\mathcal{K}}\{e^j_{l,k,1}, e^j_{l,k,2}..., e^j_{l,k,x^j_{l,k}}\}\right] \tag{4}$$

and define the (expected) job response time of job j as

$$T_j = \max_{l \in \mathcal{L}_j} \{y_l^j + e_l^j(\bar{\boldsymbol{x}}_l^j)\} - a_j,$$

where $e_{l,k,i}^j$ represents the execution time of the i-th replica of task l of job j at site k. As shown in the definitions, the response time of a task is determined by its fastest replica, and the response time of a job is determined by its slowest task. We have the expectation in (4) because $e_{l,k,i}^j$, $\forall i \in \{1, 2, ..., x_{l,k}^j\}$ depends on a random variable, as in (3).

Average Response Time: We define the average response time of the entire job set $\mathcal{J}$ as

$$\bar{T}(\mathcal{J}) = \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} T_j.$$

Problem Formulation: Having the aforementioned models, in the following, we formulate the problem of Geo-distributed Task Replication and Scheduling (Geo-TRS):

$$\begin{aligned}
\min \quad & \bar{T}(\mathcal{J}) \\
\text{s.t.} \quad & (1), (2), \\
& y_l^j \geq a_j, \forall l \in \mathcal{L}_j, \forall j, && \text{(5a)} \\
& \sum_{\forall l \in \mathcal{L}_j, \forall j: y_l^j \leq t \leq y_l^j + e_l^j} x_{l,k}^j \leq M_k, \forall k, \forall t, && \text{(5b)} \\
& y_l^j \in \mathbb{N}^+, x_{l,k}^j \in \mathbb{N}^+, \forall l \in \mathcal{L}_j, \forall j, \forall k. && \text{(5c)}
\end{aligned}$$

Constraint (5a) ensures that for any job, all of its tasks can only run after it arrives. Constraint (5b) ensures that in each site, the number of running tasks does not exceed the available computing slots at any time. Constraint (5c) specifies the decision variables' domains. We show the following observation:

Theorem 1. ***(Hardness of Geo-TRS)*** *Geo-TRS is NP-hard.*

Proof. Without considering the scaling factor in each site, the Geo-TRS problem can be simplified to the scheduling problem in [28] which has been proved to be NP-complete in the strong sense via a fairly standard reduction from 3-Partition problem. Therefore, Geo-TRS is NP-hard. □

III. The GeoClone Algorithm

We design and present a novel online algorithm, GeoClone, to determine the number of replicas for each task and schedule such replicas across different sites. GeoClone runs at each time slot and only makes decisions for *runnable* tasks, avoiding possible violations of the task dependencies. A task is runnable if all the tasks that it depends on have completed.

GeoClone consists of two steps or functions. In the first step, it estimates the number replica numbers for each runnable task based on the currently available number of slots at each site. In the second step, it selects proper sites for each task to run the replicas, achieving an expected response time threshold and allocating the slots across tasks iteratively without violating per-site slot availability.

A. Step 1: Equal-Share Replica Number Estimation

We present the algorithmic details in the function **Replica Number Estimation**. We use $\mathcal{J}(t)$ to denote the set of jobs at time slot t, and use $N(t) = |\mathcal{J}(t)|$ to denote the number of jobs in $\mathcal{J}(t)$. We introduce a parameter ε, where $0 < \varepsilon < 1$. We select the first $\lceil \varepsilon N(t) \rceil$ jobs with the least remaining tasks for execution (Line 1) at time t. For each job j inside the $\lceil \varepsilon N(t) \rceil$ jobs, on average, it can use $h_j(t)$ slots at time t (Line 3). For all the other jobs that are not in the $\lceil \varepsilon N(t) \rceil$ jobs, we set $h_j(t) = 0$ and their tasks will wait for execution in future time slots. Now, we introduce $\varpi_j(t) = \sum_k \sum_l \varpi_{l,k}^j(t)$, where $\varpi_{l,k}^j(t)$ is the number of slots that are currently occupied by task l of job j in site k at time t. The number of slots that are actually available for job j at time t can thus be calculated (Line 4). We optimistically deem that the runnable tasks should fully utilize the available slots when replication is allowed. We then equally divide a job's slot shares among all its runnable tasks (Lines 6–7), where $\sigma_j(t) = |S_j(t)|$ represents the number of runnable tasks of job j. If $c_j(t) < \sigma_j(t)$, we simply choose $c_j(t)$ tasks from the $\sigma_j(t)$ runnable tasks randomly and assign one slot to each of them. Note that, $x_l^j(t)$ can be now regarded as an *upper bound* for the number of replicas each task can have at t. Tasks with $x_l^j(t) \geq 1$ proceed to the second step.

Function Replica Number Estimation

Input: $\mathcal{J}(t)$: Set of running jobs at current time t;
$S_j(t)$, $\forall j$: Set of runnable tasks of job j at t;
M_k, $\forall k$: Number of available slots in each site;
$\varpi_j(t)$, $\forall j$: Number of slots currently occupied by tasks of job j at t;
ε: Algorithmic parameter chosen by the system operator;
Output: $x_l^j(t)$, $\forall l \in \mathcal{S}_j(t)$, $\forall j$: (Upper bound of) number of replicas for task l of job j at t;

1 $\mathcal{J}^o(t) \leftarrow$ Select the first $\lceil \varepsilon N(t) \rceil$ jobs in $\mathcal{J}(t)$ in the ascending order of the number of each job's remaining tasks;
2 **for** $j \in \mathcal{J}^o(t)$ **do**
3 $\quad h_j(t) = \left\lceil \frac{\sum_{k \in \mathcal{K}} M_k}{\varepsilon N(t)} \right\rceil$;
4 $\quad c_j(t) = \max\{h_j(t) - \varpi_j(t), 0\}$;
5 $\quad$ **for** $l \in S_j(t)$ **do**
6 $\quad\quad$ **if** $c_j(t) \geq \sigma_j(t)$ **then** $x_l^j(t) = \lfloor c_j(t)/\sigma_j(t) \rfloor$;
7 $\quad\quad$ **else** Choose $c_j(t)$ tasks from $S_j(t)$ randomly and for each chosen task l, $x_l^j(t) = 1$;

Our design is based on combining the Shortest Remaining Processing Time scheduling algorithm and the fair slot sharing approach [20], [21]. We select the "front" part of the jobs with the least remaining tasks for sharing the system-wide slots. If the number of slots allocated to a job is more than the job's current runnable tasks, extra slots are expected for hosting replicas. By carving up the slot share of the jobs (almost) equally, the number of slots allocated to each runnable task equals the number of replicas estimated. As a whole, we give more replication opportunities to the jobs which have fewer remaining tasks and get closer to the end of execution.

B. Step 2: Job-Progress-Aware Replica Placement

We present the algorithmic details in the function **Replica Placement**. At a high level, we iteratively allocate the slots to the tasks conforming to their replica upper bound. We prioritize jobs that have achieved more progress (Line 1). This can be done by checking the current stage of the job execution and the residual number of tasks in the current

stage. For tasks of the same job, each task's priority varies as the iterations (Line 7) run. In the first iteration when allocating the slots, we may have an arbitrary order of the tasks of a job. Since the second iteration, tasks are sorted in an decreasing order of the expectation of each task's current execution time $e_l^j(\bar{\boldsymbol{x}}_l^j)$, depending on the slot allocation (Lines 8–16) in previous iterations. We are doing so in order to capture that it is more urgent to spawn new replicas for tasks with worse time expectation. For each task, if it does not use out its replica number upper bound, usually, we should just directly select the currently best idle slot across all the sites and update the counters accordingly (Lines 14–16). However, an important insight from our theoretical analysis, which will be shown later, indicates that GeoClone can always have provable performance guarantees if the expected execution time of each task is maintained below a *threshold* (Line 10). We thus do not allocate any slot to a task that has already achieved an expected execution time less than the threshold (Lines 11–12). The threshold of a task is calculated using the current system utilization $u(t)$ at time t (Line 9) and the intrinsic execution time p_l^j of the task (Line 10). We stop when there is no job to serve, and for each task that has at least one replica created, we record its starting time slot (Lines 17–22).

Note that when any task l of job j has one of its replicas finish the execution, it will kill the other replicas, return all the occupied slots to the system, and update $\varpi_{l,k}^j(t) = 0, \forall k \in \mathcal{K}$.

We iteratively and "greedily" allocate slots to tasks in GeoClone, and the improvement of the execution speed of a task due to spawning a new replica gradually decreases as more replicas are created. We later formally prove this diminishing "marginal benefit" property in Lemma 1. Consequently, there are ample opportunities to serve high-priority tasks with fewer replicas without noticeable overhead; the threshold also prohibits allocating excessive replicas for a task. The saved slots can be allocated to tasks which have no or fewer replicas, achieving salient performance improvement on average.

Function Replica Placement

Input: $\mathcal{J}^o(t)$; $S_j(t)$; $\varpi_{l,k}^j(t)$; $x_l^j(t)$;
$\mathcal{K}^a(t)$: Sites that have available slots at t;
Output: $x_{l,k}^j$: Number of replicas for task l of job j in site k;
y_l^j: Starting time (for execution) of task l of job j;

1 $\mathcal{J}^a(t) \leftarrow$ Sort jobs in $\mathcal{J}^o(t)$ in the ascending order of the current job progress;
2 **while** *true* **do**
3 **for** $j \in \mathcal{J}^a(t)$ **do**
4 **if** $S_j(t)$ *is empty* **then** Remove j from $\mathcal{J}^a(t)$;
5 $S_j^o(t) = S_j(t)$;
6 /* Skip the next line in the first iteration. */
7 $S_j^o(t) \leftarrow$ Sort tasks in $S_j^o(t)$ in the descending order of each task's current expected execution time $e_l^j(\bar{\boldsymbol{x}}_l^j(t))$;
8 **for** $l \in S_j^o(t)$ **do**
9 $u(t) = \frac{\sum_{j \in \mathcal{J}(t)} \varpi_j(t)}{\sum_{k \in \mathcal{K}} M_k}$, $\alpha(t) = \frac{1}{(1+\varepsilon)u(t)}$;
10 **if** $x_l^j(t) == 0$ *or* $e_l^j(\bar{\boldsymbol{x}}_l^j(t)) \leq \frac{p_l^j}{\alpha(t)}$ **then**
11 Reject to allocate slots;
12 $S_j(t) = S_j(t) \setminus l$;
13 **else**
14 $k' = \text{argmin}_{k \in \mathcal{K}^a(t)} \mathbb{E}[e_{l,k}^j]$;
15 Allocate a slot in site k';
16 $\varpi_{l,k'}^j(t)$++, $x_l^j(t)$−−, $x_{l,k'}^j$++;
17 **if** $\mathcal{J}^a(t)$ *is empty* **then**
18 **for** $j \in \mathcal{J}^o(t)$ **do**
19 **for** $l \in S_j^o(t)$ **do**
20 **if** $\sum_{k \in \mathcal{K}} x_{l,k}^j > 0$ **then**
21 $y_l^j = t$;
22 return;

IV. Competitive Analysis

We formally prove the *competitive ratio* as the performance analysis for our algorithm. The standard competitive ratio would compare the average response time of GeoClone to that of the offline optimal algorithm; however, based on existing research [28], problems like Geo-TRS can be proven to have no bounded or finite competitive ratio. Thus, we focus on an alternative version of the competitive ratio which uses *speed augmentation* [23], [24]. We give the following definition:

Definition 1. ***(Speed-Augmented Competitiveness)*** *For a minimization problem, an online algorithm $\mathcal{A}_s(\cdot)$ is said s-speed c-competitive if it satisfies $\frac{F(\mathcal{A}_s(I))}{F(\mathcal{OPT}_1(I))} \leq c$, $\forall I$, where $F(\cdot)$ is the objective function, I is the input, $c \geq 1$ is the competitive ratio, $\mathcal{OPT}_1(\cdot)$ is the offline optimal algorithm, and $\mathcal{A}_s(\cdot)$ is the online algorithm that works in the problem setting where the resources (e.g., computing and communication) are s times as fast as those considered by the offline optimal algorithm.*

We firstly introduce a lemma as below to reveal an important insight from GeoClone, which is used in proving our speed-augmented competitive ratio: for any task of any job, compared to its current "execution speed" achieved by a number of its replicas, the improvement of its execution speed by spawning additional replicas via GeoClone is upper-bounded by the number of the additional replicas over that of the current replicas. That is, we can increase the execution speed of a task via creating more replicas; however, such improvement diminishes as the number of existing replicas increases.

Lemma 1. ***(Bounded Speed Improvement in GeoClone)*** *Let $r_l^j(\bar{\boldsymbol{x}}_l^j)$ be the expected execution speed of the fastest replica of task l of job j:*

$$r_l^j(\bar{\boldsymbol{x}}_l^j) = \mathbb{E}\left[\max_{k \in \mathcal{K}}\{r_{l,k,1}^j, r_{l,k,2}^j, ..., r_{l,k,x_{l,k}^j}^j\}\right],$$

where $r_{l,k,i}^j$, $\forall i \in \{1, 2, ..., x_{l,k}^j\}$ is the speed of the i-th replica of task l of job j at site k. For any $\bar{\boldsymbol{x}}_l^j$ and $\bar{\boldsymbol{z}}_l^j$ and any integers $0 < b \leq a$, we have $\frac{r_l^j(\bar{\boldsymbol{x}}_l^j)}{r_l^j(\bar{\boldsymbol{z}}_l^j)} \leq \frac{a}{b}$ if $x_l^j = a$ and $z_l^j = b$.

Proof. See Appendix A for details. □

We then exhibit our primary theoretical result on the speed-augmented competitive ratio of GeoClone through the following theorem—based on our algorithmic parameter ε, GeoClone

is $(1+\varepsilon)$-speed $O(\frac{1}{\varepsilon})$-competitive, and one sufficient condition for this result is that the system utilization is less than 50%. We highlight that, in reality, lots of enterprise or production clusters have been found highly underutilized. For example, Facebook's Hadoop clusters have the utilization exceed 50% for only 8% of the time [3]. That said, GeoClone works with our proven performance guarantee most of the time. Even for high-utilization clusters, we show through our evaluations in the next sections that GeoClone can practically outperform the existing replication and scheduling strategies vastly.

Theorem 2. ***(Competitive Ratio of GeoClone)*** *GeoClone is a $(1+\varepsilon)$-speed $O(\frac{1}{\varepsilon})$-competitive online algorithm with respect to the average job response time in expectation if $u(t) < \frac{1}{2}$, $\forall t$, where ε is an algorithmic parameter satisfying $0<\varepsilon<1$.*

Proof. See Appendix B for details.

Our idea for the proof is through *potential function* analysis. We construct a potential function for each time slot as the quotient of the difference between the size of the unprocessed data (to reflect the progress) of the tasks in GeoClone and that in the offline optimal algorithm, divided by the task speed in GeoClone. We differentiate the potential function—connecting to the speed and applying Lemma 1—and bound each component of the differentiation. We find the speed-augmented competitive ratio by taking integration over time.

We explain this idea very concisely as below. Let $G(t)$ and $OPT(t)$ be the cumulative job execution time at t in GeoClone and in the optimal algorithm, respectively. Let $\Psi(t)$ be the potential function, where $\Psi(0) = \Psi(\infty) = 0$. We can prove $\mathbb{E}[\frac{\mathrm{d}G(t)}{\mathrm{d}t}] + \mathbb{E}\left[\frac{\mathrm{d}\Psi(t)}{\mathrm{d}t}\right] \leq \frac{1+u}{\varepsilon(1-2u)}\mathbb{E}[\frac{\mathrm{d}OPT(t)}{\mathrm{d}t}]$. Then, we have

$$\begin{aligned}
&\int_0^\infty \mathbb{E}[\tfrac{\mathrm{d}G(t)}{\mathrm{d}t}]\mathrm{d}t + \int_0^\infty \mathbb{E}\left[\tfrac{\mathrm{d}\Psi(t)}{\mathrm{d}t}\right]\mathrm{d}t \leq \tfrac{1+u}{\varepsilon(1-2u)}\int_0^\infty \mathbb{E}[\tfrac{\mathrm{d}OPT(t)}{\mathrm{d}t}]\mathrm{d}t \\
&\Rightarrow \mathbb{E}[G] + \mathbb{E}[\Psi(\infty)] - \mathbb{E}[\Psi(0)] \leq \tfrac{1+u}{\varepsilon(1-2u)}\mathbb{E}[OPT] \\
&\Rightarrow \mathbb{E}[G] \leq \tfrac{1+u}{\varepsilon(1-2u)}\mathbb{E}[OPT].
\end{aligned}$$

For $\frac{1+u}{\varepsilon(1-2u)}$, where $u = \max_t u(t)$, to be a valid competitive ratio, we need $\frac{1+u}{\varepsilon(1-2u)} > 0$, i.e., $u(t) < 1/2$, $\forall t$. □

V. Implementation and Experiments

A. System Implementation

We implement GeoClone with Spark[1] and Yarn[2]. We implement GeoClone as a centralized service, and modify the `ResourceManager` (`RM`) in Yarn and the per-job `Driver` in Spark to coordinate the replication and scheduling of the tasks. As shown in Fig. 3, `RM` is the ultimate authority that arbitrates resources for tasks. The `Driver` of a submitted Spark job runs inside Yarn's `ApplicationMaster` (`AM`) process. `AM` negotiates resources with `RM`. Note that, instead of directly asking `RM` for the slots, the `Driver` submits its tasks to our GeoClone scheduler. GeoClone then scans the resource condition in all authorized `RM`s, and returns task placement decisions to the `Driver`. Afterwards, the `Driver` asks `RM`s for slots with admitted tokens. Inside the `Driver`,

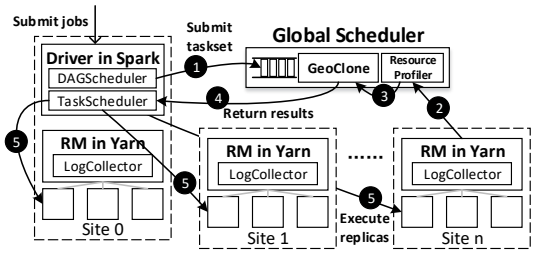


Fig. 3: Implementation of GeoClone with Spark and Yarn

`DAGScheduler` compiles the execution script into a job DAG, which encapsulates current runnable tasks into a task set and submits this task set to `TaskScheduler` for enforcing the decisions. We make the following modifications:

Submitting Tasks: We enforce `DAGScheduler` to submit its task sets to GeoClone for queuing (Step 1). The locations and the sizes of dependent data are also put into the task set, which can be obtained from the input scripts or from `Driver`.

Profiling Resources: We enforce task threads to report execution speed, used bandwidth, and site number to `AM` as they complete. We also develop the `LogCollector` service in `RM` to periodically collect such information from local `AM`s and send them to the `ResourceProfiler` module associated to GeoClone (Step 2), which parses received information and builds distributions feeding to GeoClone (Step 3).

Implementing Decisions: We trigger the algorithm when a preset timer expires. As tasks are sent to `TaskScheduler` for execution (Step 4), we entrust each task with non-local tokens according to its placements decided by GeoClone. With the tokens, tasks can be launched on a non-local slot.

Executing Tasks: We enable a task to run simultaneously in multiple slots via replicas (Step 5). Once one of these replicas finishes, we enforce the `Driver` to cancel other replicas immediately leveraging Spark's own API.

B. Experiments and Results

Testbed: We deploy our prototype on 10 Virtual Machines (VMs), each running a 64-bit Ubuntu 16.04, as in Table I. The bandwidth of each VM is set by randomly sampling from the range of 15 Mbps-50 Mbps, and enforced via Wondershaper (a Traffic Control Tool in Linux). We run (1) a set of integer and floating-point computations in Unix Benchmark Utility[3], (2) a set of memory allocations and copying operations, and (3) a set of disk read/write/seek operations in Bonnie++[4], as background workloads in each VM.

TABLE I: VM configurations

VM Role	data center	edge-1	edge-2
Number	2	6	2
Host	Dell PowerEdge r720 server	Dell PowerEdge r720 server	Dell PC
# of CPU cores	16	5 - 10	4
Size (GB) of memory	32	8 - 15	8

[1]Apache Spark, http://spark.apache.org/, 2020.

[2]Apache Yarn, https://hortonworks.com/apache/yarn/, 2020.

[3]Ubench, http://phystech.com/download/ubench.html, 2020.

[4]Bonnie++, https://www.coker.com.au/bonnie++/, 2020.

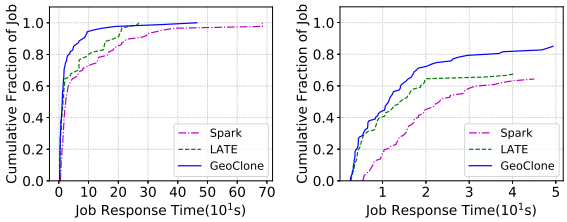
Workloads: We use *WordCount*, *iterative machine learning* and *PageRank*—88 jobs in total. The variation of the number of tasks per job is based on real-world workloads from Yahoo! and Facebook [26]. The job arrival time follows an exponential distribution, with 3 jobs per 2 minutes on average.

Baselines: We compare GeoClone, with 5-second scheduling time intervals, against the original Spark scheduler and the Spark scheduler that enables detection-based task replication. The Spark scheduler adopts fair scheduling; the detection-based task replication solution adopted in Spark is LATE [1], which speculatively spans an extra replica for an active task if its estimated progress rate is below a predefined threshold.

Metrics: We focus on the average job response time (jrt) and its cumulative distribution function (CDF).

Scheduler	Average Job Response Time
GeoClone	31.9
Spark with LATE	52.8
Spark	91.4

Fig. 4: jrt under GeoClone and Spark with/without LATE



(a) CDF of all jobs (b) CDF of jobs within 50s

Fig. 5: CDF under GeoClone and Spark with/without LATE

Results: Fig. 4 shows that GeoClone substantially cuts down the average response time in Spark and surpasses detection-based LATE by 39.6%. Fig. 5a depicts the CDF of the response time. GeoClone has a longer tail than the detection-based speculation strategy because it is impossible to prevent all task executions from straggling under GeoClone. Yet, the results still show that GeoClone benefits more from accelerating most of the jobs. Fig. 5b amplifies part of the CDF. 72.4% jobs in GeoClone finish within 20s, while the corresponding proportions in LATE and Spark are 65.6% and 45.9%, respectively.

VI. Numerical Simulations

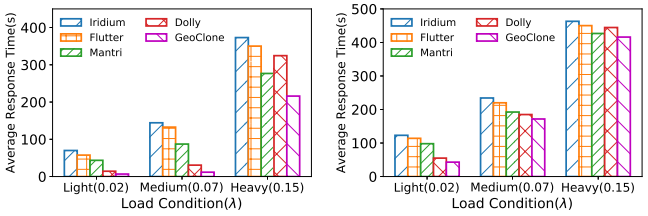
To evaluate GeoClone on a larger scale more extensively, we extend our experiments to simulations.

A. Simulation Settings

We exploit the CloudSim[5] simulator to run our simulations ranging from 20 to 500 sites. Our results are primarily from a 100-site system with 30000 slots. The number of slots for other scales is set proportionally (e.g., the 50-site system has $30000/100\times50=15000$ slots). We generate the geographical distribution of the sites via the BRITE[6] topology generator. We use the Zipf distribution to set the skewness: the higher the skew parameter is, the more skewed the slot distribution is (i.e., slots gather in fewer sites). One extreme case is that slots are distributed uniformly across all sites. The computing power of the slots is assumed to follow a normal distribution [12]. To capture heterogeneity across sites, the mean and the variance of the slot power are set based on the number of the slots in a site, and the inter-site bandwidths are set based on the distance between two sites. These values follow real-world experiments on Amazon EC2 and other clouds [1], [11], [12].

[5]CloudSim, http://www.cloudbus.org/cloudsim/, 2020.

[6]BRITE, http://www.cs.bu.edu/brite/, 2020.



(a) Facebook-like trace (b) HPC-like trace

Fig. 6: Average job response time

TABLE II: Workload statistics

Trace	Small Job (1-150)	Medium Job (150-500)	Large Job (501+)	Avg. Job Size (Task #)
Facebook-like	89%	8%	3%	90
HPC-like	0	47%	53%	430

Workloads: We synthesize workloads by Montage [29]. A job consists of the tasks with demand of both data transference and computing. In our primary results where we use the traces from Facebook's production Hadoop cluster (i.e., Facebook-like) [2], [3], [5], the distribution of the number of tasks per job has high variance with a few extremely large jobs. We also use HPC-like workloads of large jobs with small variances. See Table II. The job inter-arrival times are from a Poisson distribution, with the Poisson parameter λ in $0.02\sim0.15$.

Algorithms for Comparison: We compare GeoClone to four other algorithms: Iridium [8], Flutter [16], Flutter+Mantri [2] (Mantri for short), and Flutter+Dolly [3] (Dolly for short). Iridium and Flutter are typical geo-distributed task schedulers. The former optimizes data transference across WAN; the latter considers both computing and transference. Mantri and Dolly are the best detection-based and the best clone-based in-site replication strategy so far, respectively.

Metrics: We focus on the same metrics as in Section V-B. Further, for the three replication strategies of Dolly, Mantri and GeoClone, we study the ratio of the intensified performance by replication, in terms of the decreased jrt over the performance obtained in Flutter. We define for each job the "reduction", where "reduction $=\frac{\text{jrt in Flutter}-\text{jrt in x}}{\text{jrt in Flutter}}$" and "x" refers to the three replication strategies. We repeat every evaluation for ten times, and calculate the average response time of each job.

B. Simulation Results

Average Job Response Time: Fig. 6 depicts the average job response time for the two traces in a 100-site system. We vary the job inter-arrival time and observe how the performance

improvement reacts to different system utilization. Jobs in Iridium and Flutter (without dealing with the stragglers) far exceed their expected completion time. Dolly and Mantri mitigate the negative impact of unpredictable executions via heuristic task replication. GeoClone achieves the best performance for all workloads. It achieves more benefits for serving small jobs in the Facebook-like trace, and still works and performs the best towards large jobs in the HPC-like trace.

Distribution of Response Time Reduction: Fig. 7 shows the performance improvement through task replication in details. Fig. 7a shows that, when system utilization is light, more than 70% jobs are improved by at least 91.4% (85.2%) in GeoClone (Dolly) when compared against Flutter, while Mantri only achieves 74.3%. Mantri spends time detecting stragglers, limiting its agility of spanning replicas. Fig. 7b shows that, when system utilization is moderate, GeoClone performs the best since it exploits resource diversities and replicates tasks with high returns in job progress. Mantri and Dolly spawn replicas for their tasks regardless of resource diversities; Dolly even spawns the same number of replicas for all tasks in a job, wasting resources and damaging performance. Fig. 7c shows that, even under high system utilization, GeoClone still guarantees job response time—more than 70% jobs are improved by at least 49.6%. Mantri has its ratio of 41.1% at the 70 percentile, as it effectively restrains the overlong tasks. Dolly loses performance due to its abuse of resources, and it has no positive impact on up to 63.4% jobs.

Replica Consumption: Fig. 8 exhibits the consumption of the replicas per task via box plots with quartiles. When the system utilization is light, GeoClone consumes a moderate number of replicas (4 at the 75^{th} percentile) compared to the conservative Mantri method (2 at the 75^{th} percentile). GeoClone uses more idle resources; but compared to the aggressive Dolly method which often incurs a system utilization of greater than 50%, its utilization can be within 37% in 90% of the time. When the system becomes busy, Dolly still consumes 3 replicas per task at the 75^{th} percentile, while GeoClone reduces replicas without disturbing the normal/future job execution and improves marginal benefit of each replica.

Impact of System Heterogeneity and Scale: Fig. 9 shows that as the across-site heterogeneity increases, GeoClone outperforms other strategies more, which implies that GeoClone can handle heterogeneity efficiently. Fig. 10 shows that the performance improvement by GeoClone increases as the system scale increases, because there are more opportunities for coordinating resource usages among tasks across more sites. We also measure the communication overhead incurred by GeoClone for profiling heterogeneous resource models across sites. Fig. 12 demonstrates that such overhead is negligible.

Performance in High-Utilization Systems: We finally explore the performance of GeoClone under higher system utilizations (i.e., above 50%). Fig. 11 illustrates that, when the system utilization increases as ε takes values from Fig. 13 (which lists the corresponding best ε values as λ takes proper values leading to the system utilization beyond 50%), GeoClone has better or similar performance as Mantri.

VII. Related Work

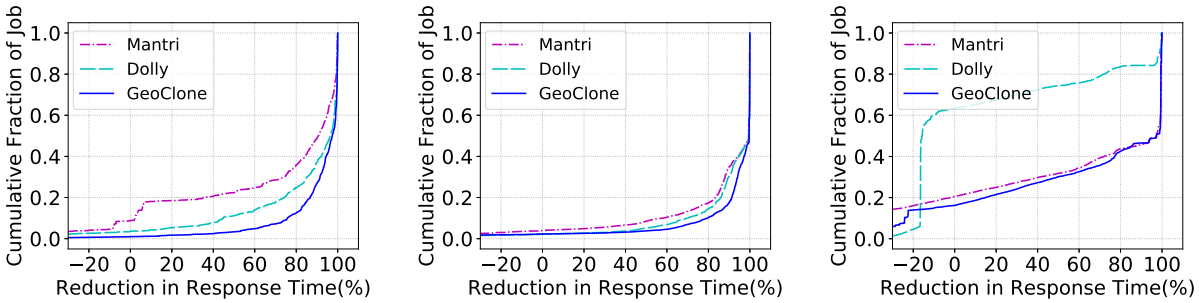
We summarize existing research in two groups, and for each group, point out their insufficiencies and limitations.

Wide-area Analytics Optimization: As the WAN is a critical bottleneck in low-latency geo-analytics, many previous researches focused on reducing the data transference delay between successive stages (e.g., map and reduce) [8], [30]–[32]. Iridium [8] migrated data across sites prior to job arrivals in order to relieve the network bottleneck during job executions. Clarinet [31] pushed the WAN-awareness up along the analytics stack into the optimization of the execution order of the queries in order to lower the response time. Sana [32] allowed queries to share common executions, and eliminated redundant data transference and processing. Other researches also optimized the computing delay via carefully scheduling the jobs, given the heterogeneities in the computing capacities of clouds and edges [9], [10], [14]–[17]. All such existing works largely assume constant capacities of the resources. However, due to common component failures in these exascale systems [33], [34] and contention on the resources from other non-analytics jobs, the available capacities of both WAN and computing slots can fluctuate significantly over time [11]–[13].

Task Replication for Stragglers: Task replication has been a well-adopted technique to tackle stragglers in data-parallel processing systems. One important type of replication strategies is *detection-based*, i.e., first detecting stragglers and then replicating tasks accordingly [1], [2], [4], [5]. Mantri [2] considered the opportunity cost using extra slots for replicas, and spawns new replicas only if both time and resources could be saved compared to original executions. GRASS [5] focused on time-saving-only replication, since the opportunity cost proved to diminish for jobs with fewer unscheduled tasks which could benefit the job more as it got closer to completion. Detection of stragglers actually relies on spending time comparing the performance of other tasks of the same job, which often limits the efficacy of using replicas for speeding up tasks, especially unsuitable for small jobs. Another type of replication strategies is *clone-based*, i.e., launching multiple replicas for tasks just as scheduled according to predictions of stragglers in historical executions [3], [6], [7]. However, task clone consumes extra resources more aggressively, which can adversely impact the resource availability of normal job executions. To alleviate this negative impact, [3] cloned small jobs without significantly increasing the overall cluster utilization. [6], [7] further coordinated the resource usage between clones and the original executions within clusters. Such existing in-site cloning algorithms assume that replicas can run anywhere, with nearly homogeneous slot capacity and abundant intra-site networks. These assumptions are, however, incompatible with geo-distributed cloud-edge systems, where, besides the number of replicas, the locations of replicas also impact the completion of the tasks due to resource heterogeneities.

VIII. Conclusion

In this paper, we are the first to investigate task replication for a cloud-edge data-parallel system. We design GeoClone,



(a) Light load ($\lambda = 0.02$) (b) Medium load ($\lambda = 0.07$) (c) Heavy load ($\lambda = 0.15$)

Fig. 7: CDF of job response time reduction

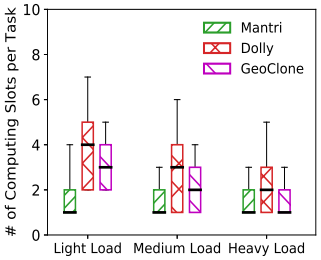


Fig. 8: Replica # per task

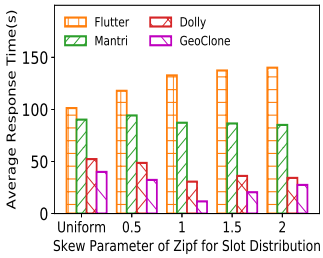


Fig. 9: Impact of heterogeneity

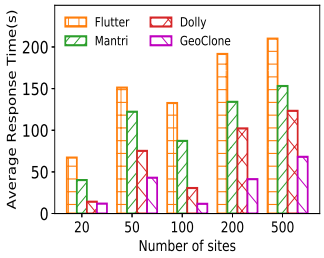


Fig. 10: Impact of scale

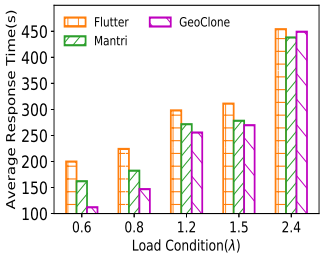


Fig. 11: Under high utilization

Site size	20	50	100	200	500
Overhead (%)	0.46	0.81	1.93	3.88	7.26

Fig. 12: Communication overhead

λ	0.6	0.8	1.2	1.5	2.4
best ε	0.6	0.6	0.8	0.8	0.9

Fig. 13: Best ε for high-utilization λ

an online scheduling algorithm for geo-distributed job replication and execution. Based on our theoretical insight that the opportunity of using task replicas to improve the overall job performance diminishes as the system goes busy, we selectively replicate a subset of jobs by dynamically weighing the replica-incurred benefits in the job progress and adapting to the system utilization. We formally prove our approach's performance against the offline optimum. Our implementations and simulations confirm that GeoClone drastically improves the job performance in practice and outperforms the current best cluster-scale replication and scheduling strategies.

ACKNOWLEDGEMENT

This research is partially supported by the National Key R&D Program of China (Grant No. 2017YFB1001801), the Natural Science Foundation of China (Grant No. 61832005, 61802182), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

APPENDIX

A. Proof of Lemma 1

Proof. When task l has n replicas in the system, its execution speed can be denoted as $r_n = \max\{v_1, v_2, \cdots, v_x, \cdots, v_n\}$, where v_x is the execution speed of its x-th replica and follows a distribution, i.e., $v_x \sim F_x(v) \triangleq Pr(v_x < v)$. Define $Q_n(v)$ as the cumulative distribution function of r_n. Then, we have

$$Q_n(v) = \prod_{x=1}^{n} F_x(v)$$

following the definition of cumulative distribution functions. Next, define $q_n(v) \triangleq \mathrm{d}(Q_n(v))/\mathrm{d}v$ and $f_x(v) \triangleq \mathrm{d}(F_x(v))/\mathrm{d}v$. Then, we have

$$q_n(v) = \sum_{x=1}^{n} \left(f_x(v) \cdot \prod_{\substack{j=1:n;\\ j\neq x}} F_j(v) \right).$$

Now, let us prove

$$n\mathbb{E}[r_{n+1}] \leq (n+1)\mathbb{E}[r_n], \forall n \geq 1. \tag{6}$$

We expand the left-hand side of Eq. (6) as follows, based on the definition of the expectation of random variables:

$$n\mathbb{E}[r_{n+1}] = n\left(\int v \cdot q_{n+1}(v)\mathrm{d}v\right) = n\left(\int v \cdot \sum_{x=1}^{n+1}\left(f_x(v) \prod_{\substack{j=1:n+1;\\ j\neq x}} F_j(v)\right)\mathrm{d}v\right)$$

$$= n\left(\int v \cdot \sum_{x=1}^{n}\left(f_x(v) \prod_{\substack{j=1:n+1;\\ j\neq x}} F_j(v)\right)\mathrm{d}v + \int v \cdot f_{n+1}(v) \prod_{j=1}^{n} F_j(v)\mathrm{d}v\right)$$

$$= n\int v \cdot q_n(v) F_{n+1}(v)\mathrm{d}v + n\int v \cdot f_{n+1}(v) \prod_{j=1}^{n} F_j(v)\mathrm{d}v. \tag{7}$$

Because we have $F_{n+1}(v) \leq 1$ by definition, then

$$(7) \leq n\int v \cdot q_n(v)\mathrm{d}v + n\int v \cdot f_{n+1}(v) \prod_{j=1}^{n} F_j(v)\mathrm{d}v$$
$$= n \cdot \mathbb{E}[r_n] + n\int v \cdot f_{n+1}(v) \prod_{j=1}^{n} F_j(v)\mathrm{d}v. \tag{8}$$

Recall that, for the newest replica of a task, GeoClone greedily selects the *current* best slot (i.e., with the highest expected speed) across all sites that have available slots. Thus, we have

$$\mathbb{E}[v_{n+1}] \leq \mathbb{E}[v_x], \ \forall x < n+1, \tag{9}$$

since a replica receives a better slot if GeoClone allocates it earlier. Based on Eq. (9), we have

$$n\mathbb{E}[v_{n+1}] \leq \sum_{x=1}^{n} \mathbb{E}[v_x]$$
$$\Rightarrow n\int v f_{n+1}(v)\mathrm{d}v \leq \sum_{x=1}^{n} \int v \cdot f_x(v)\mathrm{d}v$$

$$\Rightarrow n\int v f_{n+1}(v)\prod_{j=1}^{n}F_j(v)\mathrm{d}v \le \int v\cdot\sum_{x=1}^{n}\left(f_x(v)\cdot\prod_{\substack{j=1;n+1;\\ j\neq x}}F_j(v)\right)\mathrm{d}v$$

$$\Rightarrow n\int v f_{n+1}(v)\prod_{j=1}^{n}F_j(v)\mathrm{d}v \le \mathbb{E}\left[r_n\right]. \tag{10}$$

Putting Eq. (10) into Eq. (8), we can prove Eq. (6), based on which, for any integers $a \ge b > 0$, we have

$$\frac{\mathbb{E}[r_b]}{b} \ge \frac{\mathbb{E}[r_{b+1}]}{b+1} \ge \cdots \ge \frac{\mathbb{E}[r_a]}{a}.$$

The proof completes. □

B. Proof of Theorem 2

Proof. We start with defining a special *potential function*. Let $z_l^j(t) \triangleq \max(d_l^{jG}(t) - d_l^{jO}(t), 0)$, where $d_l^{jG}(t)$ and $d_l^{jO}(t)$ represent the volume or the size of the remaining unprocessed data of task l of job j at time t under the optimal algorithm and the GeoClone algorithm, respectively. Now, we define the potential function for a single task as

$$\varphi_l^j(t) = \frac{z_l^j(t)}{r_l^j(M_{\mathcal{K}}/\varepsilon N(t))},$$

where $M_{\mathcal{K}} = \sum_{k\in\mathcal{K}} M_k$, and define the potential function for all the jobs as

$$\Psi(t) = \frac{1}{\varepsilon^2}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j}\varphi_l^j(t),$$

where $\eta^G(t)$ is the set of the running jobs at t in GeoClone. Similarly, we let $\eta^O(t)$ and $\eta_l^O(t)$ denote the set of running jobs and the set of running tasks at t in the optimal scheduling, respectively. Note, since GeoClone can be different from the optimal algorithm, the sets of running tasks and jobs at t can be different for GeoClone and for the optimum.

We proceed with the differentiation of the potential function. We have

$$\mathbb{E}\left[\frac{d\Psi(t)}{dt}\right] = \frac{1}{\varepsilon^2}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j}\mathbb{E}\left[\frac{d\varphi_l^j(t)}{dt}\right].$$

We can also represent or decompose this differentiation as

$$\mathbb{E}\left[\frac{\mathrm{d}\Psi(t)}{\mathrm{d}t}\right] = \Delta^O(t) + \Delta^G(t) + \Delta^D(t),$$

where $\Delta^O(t)$ refers to the contribution to the differentiation by the optimal scheduling, $\Delta^G(t)$ refers to the contribution to the differentiation by GeoClone, and $\Delta^D(t)$ refers to the contribution to the differentiation by job arrivals and completions.

In the following, we conduct the three steps of bounding $\Delta^O(t)$ and $\Delta^G(t)$, respectively, and deriving the competitive ratio via integrating $\mathbb{E}\left[\frac{\mathrm{d}\Psi(t)}{\mathrm{d}t}\right]$ over time. Before proceeding to the three steps, we point out the following, which will be used in our derivations. For job j, let f_j^G be its completion time in GeoClone; for time $t \ge a_j$, let $G_j(t) = \min(f_j^G, t) - a_j$ be the cumulative execution time of job j until time t. We have

$$\mathbb{E}[\tfrac{\mathrm{d}G_j(t)}{\mathrm{d}t}] = 1, \quad a_j < t < f_j^G. \tag{11}$$

Further, let $G(t) = \sum_j G_j(t)$, and $G = \sum_j G_j(\infty)$ which denotes the total job response time for GeoClone. Let $OPT_j(t)$, $OPT(t)$, and OPT be defined analogously for the optimal algorithm. Now, we perform the three steps.

Step 1: Bounding $\Delta^O(t)$. We define $\Delta_l^{jO}(t) = \frac{\mathrm{d}\mathbb{E}[\varphi_l^j(t)]}{\mathrm{d}t}$ for task l of job j at time t. Then, based on the definition of $\varphi_l^j(t)$, we have

$$\Delta_l^{jO}(t) \le -\frac{\mathbb{E}[\frac{\mathrm{d}(d_l^{jO}(t))}{\mathrm{d}t}]}{r_l^j(M_{\mathcal{K}}/\varepsilon N(t))} = \frac{s_l^j(u_l^{jO})}{r_l^j(M_{\mathcal{K}}/\varepsilon N(t))}. \tag{12}$$

The equality in Eq. (12) is as we introduce the speed $s_l^j(\cdot)$ of task l of job j achieved by the optimal algorithm for the u_l^{jO} replicas created by the optimal algorithm. We assume $s_l^j(\cdot)$ is a concave and nondecreasing function. Then, note that, due to Line 10 of **Replica Placement** in GeoClone, we have

$$r_l^j(M_{\mathcal{K}}/\varepsilon N(t)) \ge \alpha(t)\cdot q_l^j \ge \alpha(t)\cdot s_l^j(M_{\mathcal{K}}/\varepsilon N(t)), \tag{13}$$

where we use q_l^j to denote the "intrinsic" execution speed of task l of job j, similar to p_l^j which is the intrinsic execution time as described previously. $q_l^j \ge s_l^j(\cdot)$ holds, because no scheduling algorithm can achieve a greater speed than the intrinsic. Now, putting Eq. (13) into (12) yields

$$\Delta_l^{jO}(t) \le \frac{s_l^j(u_l^{jO})}{r_l^j(M_{\mathcal{K}}/\varepsilon N(t))} \le \frac{s_l^j(u_l^{jO})}{\alpha\cdot s_l^j(M_{\mathcal{K}}/\varepsilon N(t))},$$

where $\alpha = \min_t \alpha(t)$. Consider two cases. In the first case, when $u_l^{jO} \le M_{\mathcal{K}}/\varepsilon N(t)$, then we have $\Delta_l^{jO} \le 1/\alpha$ due to the monotonic property of the $s_l^j(\cdot)$ function. In the second case, when $u_l^{jO} > M_{\mathcal{K}}/\varepsilon N(t)$, then we have

$$\Delta_l^{jO}(t) \le \frac{s_l^j(u_l^{jO})}{\alpha\cdot s_l^j(M_{\mathcal{K}}/\varepsilon N(t))} \le \frac{u_l^{jO}}{\alpha\cdot M_{\mathcal{K}}/\varepsilon N(t)} = \frac{\varepsilon N(t) u_l^{jO}}{\alpha M_{\mathcal{K}}}. \tag{14}$$

This is because, if $s_l^j(\cdot)$ is concave, then it means

$$\lambda s_l^j(\beta) \le s_l^j(\lambda\beta + (1-\lambda)\alpha) - (1-\lambda)s_l^j(\alpha)$$

for $0 \le \lambda \le 1$. We reach Eq. (14) by setting $\alpha = 0$, $\beta = u_l^{jO}$, $\lambda = \frac{M_{\mathcal{K}}/\varepsilon N(t)}{u_l^{jO}}$. Summarizing the two cases, we can have

$$\Delta_l^{jO}(t) \le 1/\alpha + \frac{\varepsilon N(t) u_l^{jO}}{\alpha M_{\mathcal{K}}}. \tag{15}$$

Based on Eq. (11), it follows that

$$\begin{aligned}
\Delta^O(t) &= \tfrac{1}{\varepsilon^2}\sum_{j\in\eta^G(t)\cap\eta^O(t)}\sum_{l\in\mathcal{L}_j}\Delta_l^{jO}(t)\\
&\le \tfrac{1}{\alpha\varepsilon^2}\sum_{j\in\eta^O(t)}\sum_{l\in\mathcal{L}_j} 1 + \tfrac{N(t)}{\alpha\varepsilon M_{\mathcal{K}}}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j} u_l^{iO}\\
&\le \tfrac{1}{\alpha\varepsilon^2}\textstyle\sum_{j\in\eta^O(t)}\sum_{l\in\mathcal{L}_j} 1 + \tfrac{N(t)}{\alpha\varepsilon M_{\mathcal{K}}} M_{\mathcal{K}}\\
&\le \tfrac{1}{\alpha\varepsilon^2}\textstyle\sum_{j\in\eta^O(t)}\sum_{l\in\mathcal{L}_j} 1 + \tfrac{1}{\alpha\varepsilon}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j} 1.
\end{aligned} \tag{16}$$

Using Eq. (11) in Eq. (16), we eventually have

$$\begin{aligned}
\Delta^O(t) &\le \tfrac{1}{\alpha\varepsilon^2}\textstyle\sum_{j\in\eta^O(t)}\mathbb{E}[\frac{\mathrm{d}OPT_j(t)}{\mathrm{d}t}] + \frac{1}{\alpha\varepsilon}\sum_{j\in\eta^G(t)}\mathbb{E}[\frac{\mathrm{d}G_j(t)}{\mathrm{d}t}]\\
&= \tfrac{1}{\alpha\varepsilon^2}\mathbb{E}[\tfrac{\mathrm{d}OPT(t)}{\mathrm{d}t}] + \tfrac{1}{\alpha\varepsilon}\mathbb{E}[\tfrac{\mathrm{d}G(t)}{\mathrm{d}t}].
\end{aligned} \tag{17}$$

Step 2: Bounding $\Delta^G(t)$, with $1+\varepsilon$ speed augmentation. Let u_l^{jG} be the number of replicas created for task l of job j in GeoClone at time t. With the $1+\varepsilon$ multiplication, we have

$$\begin{aligned}
\Delta^G(t) &\le (1+\varepsilon)\cdot\tfrac{1}{\varepsilon^2}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j\cap l\notin\eta_l^O(t)}\frac{\mathbb{E}[\frac{\mathrm{d}(d_l^{jG}(t))}{\mathrm{d}t}]}{r_l^j(M_{\mathcal{K}}/\varepsilon N(t))}\\
&= -\tfrac{1+\varepsilon}{\varepsilon^2}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j\cap l\notin\eta_l^O(t)}\frac{r_l^j(u_l^{jG})}{r_l^j(M_{\mathcal{K}}/\varepsilon N(t))}.
\end{aligned} \tag{18}$$

Based on Lemma 1, as $u_l^{jG} \leq M_{\mathcal{K}}/\varepsilon N(t)$, we have

$$(18) \leq -\frac{(1+\varepsilon)N(t)}{\varepsilon}\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j\cap l\notin\eta_l^O(t)}\frac{u_l^{jG}}{M_{\mathcal{K}}}$$
$$\leq -\frac{(1+\varepsilon)N(t)}{\varepsilon}\Big(\frac{\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j}u_l^{jG}}{M_{\mathcal{K}}}-\sum_{l\in\eta_l^O(t)}\frac{u_l^{jG}}{M_{\mathcal{K}}}\Big). \quad (19)$$

Due to Line 3 of **Replica Number Estimation** in GeoClone, we have

$$\sum_{l\in\mathcal{L}_j}u_l^{jG}=h_j(t). \quad (20)$$

Based on Eq. (20),

$$\sum_{j\in\eta^G(t)}\sum_{l\in\mathcal{L}_j}u_l^{jG}=\sum_{j\in\eta^G(t)}h_j(t)\geq(1-u(t))M_{\mathcal{K}}, \quad (21)$$

where $u(t)$ is the global slot utilization at time t. Meanwhile,

$$\sum_{l\in\eta_l^O(t)}u_l^{jG}\leq\sum_{j\in\eta^O(t)}\sum_{l\in\mathcal{L}_j}u_l^{jG}=\sum_{j\in\eta^O(t)}\frac{M_{\mathcal{K}}}{\varepsilon N(t)}. \quad (22)$$

Putting Eq. (21) and Eq. (22) into Eq. (19), and also using Eq. (11), we have

$$\Delta^G(t)\leq-\frac{(1+\varepsilon)N(t)}{\varepsilon}\Big(\frac{(1-u(t))M_{\mathcal{K}}}{M_{\mathcal{K}}}-\frac{\sum_{j\in\eta^O(t)}\frac{M_{\mathcal{K}}}{\varepsilon N(t)}}{M_{\mathcal{K}}}\Big)$$
$$=-\frac{(1+\varepsilon)(1-u(t))}{\varepsilon}N(t)+\frac{1+\varepsilon}{\varepsilon^2}\sum_{j\in\eta^O(t)}1$$
$$\leq-\frac{(1+\varepsilon)(1-u)}{\varepsilon}\sum_{j\in\eta^G(t)}1+\frac{1+\varepsilon}{\varepsilon^2}\sum_{j\in\eta^O(t)}1$$
$$=-\frac{(1+\varepsilon)(1-u)}{\varepsilon}\mathbb{E}[\tfrac{\mathrm{d}G(t)}{\mathrm{d}t}]+\frac{1+\varepsilon}{\varepsilon^2}\mathbb{E}[\tfrac{\mathrm{d}OPT(t)}{\mathrm{d}t}], \quad (23)$$

where $u=\max_t u(t)$.

Step 3: Deriving the competitive ratio using integration. Integrating the changes over time, we have

$$\int_0^\infty\mathbb{E}\left[\tfrac{\mathrm{d}\Psi(t)}{\mathrm{d}t}\right]\mathrm{d}t=\int_0^\infty(\Delta^D(t)+\Delta^O(t)+\Delta^G(t))\mathrm{d}t$$

Note the facts of $\int_0^\infty\mathbb{E}\left[\frac{\mathrm{d}\Psi(t)}{\mathrm{d}t}\right]\mathrm{d}t=\mathbb{E}[\Psi(\infty)]-\mathbb{E}[\Psi(0)]=0$ and $\Delta^D(t)\leq 0$. We have $\Psi(0)=\Psi(\infty)=0$, because before the time horizon starts and after a sufficiently long time, there exists no unprocessed data and no job in the system; we have $\Delta^D(t)\leq 0$, because job arrivals increase the amount of the unprocessed data while job completions do not influence the amount of the unprocessed data. Thus, we have

$$-\int_0^\infty\Delta^G(t)\mathrm{d}t\leq\int_0^\infty\Delta^O(t)\mathrm{d}t. \quad (24)$$

Putting Eq. (17) and Eq. (23) into Eq. (24), it follows that

$$\tfrac{\alpha(1+\varepsilon)(1-u)-1}{\alpha\varepsilon}\int_0^\infty\mathbb{E}[\tfrac{\mathrm{d}G(t)}{\mathrm{d}t}]\mathrm{d}t\leq\tfrac{\alpha(1+\varepsilon)+1}{\alpha\varepsilon^2}\int_0^\infty\mathbb{E}[\tfrac{\mathrm{d}OPT(t)}{\mathrm{d}t}]\mathrm{d}t$$
$$\Rightarrow\int_0^\infty\mathbb{E}[\tfrac{\mathrm{d}G(t)}{\mathrm{d}t}]\mathrm{d}t\leq\tfrac{\alpha(1+\varepsilon)+1}{\alpha\varepsilon(1+\varepsilon)(1-u)-\varepsilon}\int_0^\infty\mathbb{E}[\tfrac{\mathrm{d}OPT(t)}{\mathrm{d}t}]\mathrm{d}t$$
$$\Rightarrow\mathbb{E}[G]\leq\tfrac{\alpha(1+\varepsilon)+1}{\alpha\varepsilon(1+\varepsilon)(1-u)-\varepsilon}\mathbb{E}[OPT]$$
$$\Rightarrow\mathbb{E}[G]\leq\tfrac{1+u}{\varepsilon(1-2u)}\mathbb{E}[OPT].$$

The last line above is because $\alpha=\min_t\alpha(t)$, $\alpha(t)=\frac{1}{(1+\varepsilon)u(t)}$, and $u=\max_t u(t)$. For $\frac{1+u}{\varepsilon(1-2u)}$ to be a valid competitive ratio, we need $\frac{1+u}{\varepsilon(1-2u)}>0$, i.e., $u(t)<1/2$, $\forall t$. □

REFERENCES

[1] M. Zaharia, A. Konwinski, A. D. Joseph *et al.*, "Improving mapreduce performance in heterogeneous environments," in *USENIX OSDI*, 2008.

[2] G. Ananthanarayanan, S. Kandula, A. Greenberg *et al.*, "Reining in the outliers in map-reduce clusters using mantri," in *USENIX OSDI*, 2010.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker *et al.*, "Effective straggler mitigation: Attack of the clones," in *USENIX NSDI*, 2013.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] G. Ananthanarayanan, C. Hung, X. Ren *et al.*, "Grass: Trimming stragglers in approximation analytics," in *USENIX NSDI*, 2014.

[6] X. Ren, G. Ananthanarayanan *et al.*, "Hopper: Decentralized speculation-aware cluster scheduling at scale," in *ACM SIGCOMM*, 2015.

[7] H. Xu and W. C. Lau, "Task cloning algorithms in a mapreduce cluster with competitive performance bounds," in *IEEE ICDCS*, 2015.

[8] Q. Pu, G. Ananthanarayanan, P. Bodik *et al.*, "Low latency geo-distributed data analytics," in *ACM SIGCOMM*, 2015.

[9] L. Jiao, L. Pu, L. Wang, X. Lin, and J. Li, "Multiple granularity online control of cloudlet networks for edge computing," in *IEEE SECON*, 2018.

[10] W. You, L. Jiao, S. Bhattacharya, and Y. Zhang, "Dynamic distributed edge resource provisioning via online learning across timescales," in *IEEE SECON*, 2020.

[11] J. Dejun, G. Pierre, and C. H. Chi, "EC2 performance analysis for resource provisioning of service-oriented applications," in *ICSOC*, 2010.

[12] J. Schad, J. Dittrich, and J. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," in *VLDB*, 2010.

[13] X. Zhang, Z. Qian, S. Zhang *et al.*, "Towards reliable (and efficient) job executions in a practical geo-distributed data analytics system," *arXiv preprint arXiv:1802.00245*, 2018.

[14] C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *ACM SoCC*, 2015.

[15] H. Tan, Z. Han, X. Li *et al.*, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM*, 2017.

[16] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *IEEE INFOCOM*, 2016.

[17] C. Hung, G. Ananthanarayanan, L. Golubchik *et al.*, "Wide-area analytics with multiple resources," in *ACM EuroSys*, 2018.

[18] H. Xu and W. C. Lau, "Optimization for speculative execution in a mapreduce-like cluster," in *IEEE INFOCOM*, 2015.

[19] K. Gardner, M. Harchol-Balter *et al.*, "A better model for job redundancy: Decoupling server slowdown and job size," *IEEE/ACM transactions on Networking*, vol. 25, no. 6, pp. 3353–3367, 2017.

[20] K. Fox, S. Im, and B. Moseley, "Energy efficient scheduling of parallelizable jobs," in *ACM-SIAM SODA*, 2013.

[21] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs, "Scheduling jobs with varying parallelizability to reduce variance," in *ACM SPAA*, 2010.

[22] H. Xu and W. Lau, "Speculative execution for a single job in a mapreduce-like system," in *IEEE CLOUD*, 2014.

[23] P. Berman and C. Coulston, "Speed is more powerful than clairvoyance," *Nordic Journal of Computing*, vol. 6, no. 2, pp. 181–193, 1999.

[24] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *Journal of the ACM*, vol. 47, no. 4, pp. 214–221, 2000.

[25] M. Zaharia, M. Chowdhury, T. Das *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *USENIX NSDI*, 2012.

[26] V. K. Vavilapalli *et al.*, "Apache Hadoop Yarn: Yet another resource negotiator," in *ACM SoCC*, 2013.

[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[28] Y. Zheng, N. B. Shroff *et al.*, "A new analytical technique for designing provably efficient mapreduce schedulers," in *IEEE INFOCOM*, 2013.

[29] G. B. Berriman *et al.*, "Montage:A grid-enabled engine for delivering custom science-grade mosaics on demand," in *SPIE*, 2004.

[30] A. Vulimiri, C. Curino, P. B. Godfrey *et al.*, "Global analytics in the face of bandwidth and regulatory constraints," in *USENIX NSDI*, 2015.

[31] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in *USENIX OSDI*, 2016.

[32] A. Jonathan, A. Chandra, and J. Weissman, "Multi-query optimization in wide-area streaming analytics," in *ACM SoCC*, 2018.

[33] X. Chen, C. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A google cluster case study," in *IEEE ISSRE*, 2014.

[34] A. Rosa, L. Chen, and W. Binder, "Understanding the dark side of big data clusters: An analysis beyond failures," in *IEEE/IFIP DSN*, 2015.