A Foundation for Real-time Applications on Function-as-a-Service

Hai Duc Nguyen
University of Chicago
ndhai@cs.uchicago.edu

Andrew A. Chien
University of Chicago & Argonne National Laboratory
aachien@cs.uchicago.edu

ABSTRACT

Serverless (or Function-as-a-Service) compute model enables new applications with dynamic scaling. However, all current Serverless systems are best-effort, and as we prove this means they cannot guarantee hard real-time deadlines, rendering them unsuitable for such real-time applications.

We analyze a proposed extension of the Serverless model that adds a guaranteed invocation rate to the serverless model called Real-time Serverless. This approach aims to meet real-time deadlines with dynamically allocated function invocations. We first prove that the Serverless model does not support real-time guarantees. Next, we analyze Real-time Serverless, showing it can guarantee application real-time deadlines for rate-monotonic real-time workloads. Further, we derive bounds on the required invocation rate to meet any set of workload runtimes and periods. Subsequently, we explore an application technique, pre-invocation, and show that it can reduce the required guaranteed invocation rate. We derive bounds for the feasible rate guarantee reduction, and corresponding overhead in wasted compute resources. Finally, we apply the theoretical results to improve the experience quality of a distributed virtual reality/augmented reality application as well as simplify the application design and resource management.

Keywords

Real-time, Serverless, FaaS, Rate-monotonic

1. INTRODUCTION

Serverless has seen a huge growth in usage [30] and received much attention from the research community [3,14]. The core technology behind serverless computation is *stateless functions* written to perform specific tasks without server deployment and management. Users associate functions with events (e.g., image upload). When such an event happens, the function logic starts execution in response (e.g., resize the image). Current serverless systems (e.g., AWS Lambda [2], Google Cloud Functions [15], etc.) support simultaneous events enabling these functions to scale up to thousands of parallel invocations.

The serverless compute model is appealing to many applications. First, serverless leverages dynamic scaling that gives no resources to a serverless function until it is triggered by an associated event. Once triggered, the function gets resources proportional to its invocation concurrency. After these invocations finish, the allocated resources are reclaimed, resulting in efficient resource utilization. Second, serverless decouples application logic from underlying resource configuration. Once a serverless function deploys, it can execute anywhere independent of the underlying resource configurations, without any additional actions from the applications. This brings great flexibility allowing one to implement complicated applications on serverless with minimum effort (e.g., [8]).

Inspired by these advantages, we seek to extend Serverless to applications with hard real-time guarantees. To date, real-time applications have been built on static resource allocation (e.g., dedicated machines) that provide continuous, static resource configuration (e.g., stable CPU speed, non-sharing memory, etc.) required by real-time scheduling and resource management to meet hard real-time deadlines. Resources required by real-time scheduling and resource management depend on underlying resource configurations [19]. For example, allowing real-time tasks to be preemptible makes scheduling decisions more flexible and therefore requires fewer resources. Such dependency ties real-time applications to their execution environment so nontrivial efforts are required when they need to expand or upgrade to use different, heterogeneous, or distributed hardware.

With dynamic allocation and high portability, implementing real-time applications over serverless can resolve the above issues straightforwardly. However, due to the lack of analytical study on implementing real-time applications with dynamic allocation resource models such as serverless, limited understandings of real-time performance, cost, and scheduling are provided. This prevents us from efficiently exploiting serverless advantages for real-time applications.

In this paper, we develop a theoretical framework by extending the rate-monotonic real-time workload model to the case of dynamic task execution atop serverless resource provisioning. By utilizing the widely-studied, well-understood foundation of the rate-monotonic framework [19], we perform execution analysis of real-time tasks and reveal that *current serverless systems have no ability to support hard real-time applications* despite the widespread marketing of real-time capabilities on serverless cloud services as they are largely *best-effort, online streaming* services.[1]

The analysis shows that the unbounded latency of the best-effort serverless allocation is a key challenge for real-time implementation. Real-time serverless [27] is one approach to resolve this issue by extending the serverless inter-

Copyright is held by author/owner(s).

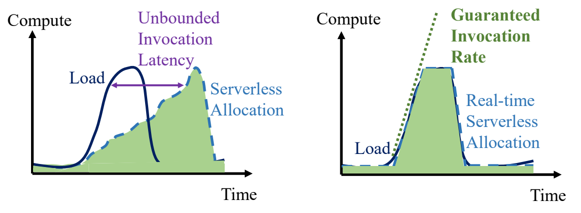
[1] For example, see [38].

face with an invocation rate guarantee and shows empirically its attractiveness for bursty, real-time applications. Benefits include enabling applications to tune quality, response time, and demonstration of effective statistical multiplexing on the servers. However, no rigorous proof of the application's ability to achieve hard real-time deadline guarantees based on the real-time serverless interface was given. Thus, we use our analytical framework to formally analyze the *real-time serverless* model to characterize its power to enable application hard real-time guarantees and the resource costs associated with such guarantees. First, we study the real-time serverless, consider the effect of its invocation rate guarantees, and show the rate-guarantee can ensure hard real-time deadlines and the required invocation rates. This is notable because hard real-time guarantees can be achieved without wasted application resources. Second, we consider an application technique, pre-invocation, that trades some application resource waste for a reduction in the required guaranteed invocation rate. Finally, we apply insights from the analytical results to illustrate how applications can use real-time serverless to satisfy hard real-time constraints through a practical distributed virtual/augmented reality case study.

Given n rate-monotonic tasks, each has A_i as its guaranteed invocation rate, and p_i, r_i, s_i are the task period, runtime, and slack (e.g., $s_i = p_i - r_i$), respectively. Specific contributions of the paper include:

- Show that current Serverless system implementation cannot provide hard real-time guarantees.
- Proof that the addition of an invocation rate guarantee enables *real-time serverless* to guarantee hard real-time performance for all rate-monotonic applications.
- Proof that assigning the inverse of the task slack (i.e., period - runtime) as the guaranteed invocation rate for each task fulfills the real-time requirements for an arbitrary multi-task rate monotonic application. Thus ($A_i = \frac{1}{s_i} = \frac{1}{p_i - r_i}$) holds for each task.
- An approach to use pre-invocation by real-time serverless applications that reduces the required guaranteed total invocation to $A_{total} - \sum_{i=1}^{n} \frac{1}{p_i}$ with the pre-invocation overhead of at most 100% (wasted resources).
- Demonstration of serving a demanding distributed virtual/augmented reality application with real-time serverless that validates the theoretical results and reveals their implications that open new capabilities to simplify application design and resource management.

The remainder of the paper is organized as follows. In Section 2, we give background on the serverless compute model and the rate-monotonic real-time model. Subsequently, in Sections 3 and 4 we frame the opportunity and describe the analytical framework. Next, in Sections 5 and 6 we prove key limitations and properties for Serverless and Real-time Serverless models. Following that, Section 7 explores the benefits and limits of pre-invocation. Next, in Section 8 we show how the analytical insights can be applied in practice. Finally, Sections 9 and 10 place our work in context and suggest future directions.



(a) Serverless (best-effort) (b) Real-time Serverless

Figure 1: Best-effort vs. Rate-guarantee Allocation [27]

2. BACKGROUND

2.1 Serverless Compute Model

Since the introduction of AWS Lambda [2], serverless computing has experienced rapid growth in usage [13] and was widely adopted by not only cloud services but also the Edge [9, 16, 33, 37] and HPC [8, 36, 41]. The serverless compute model lets applications be composed as a set of stateless functions, each often consisting of a few lines of code, running short tasks, and lasting from milliseconds to minutes. These functions are registered in a serverless system (e.g. AWS Lambda [2], OpenFaaS [29], etc.), and triggered by some events (e.g., data creation, executing a Web API, etc.) defined by the function developers. There is no specific resource allocation tied to a serverless function until an associated event triggers an invocation to start. At this moment, the serverless system dynamically finds a set of resources that satisfy the function requirement, uses these resources to create an appropriate execution environment, and starts the function logic.

State-of-the-art serverless systems demonstrate that dynamic allocation could support thousands of concurrent invocations of a single function within a few seconds. This makes serverless an intriguing match for short, periodic tasks which are frequently found in real-time systems. However, all of the current serverless systems provide no performance SLOs – even a single invocation can take an unbounded time to start, or can even be canceled. Thus, it is impossible for serverless to provide guarantees of real-time performance, as shown in Figure 1a (purple line). Unbounded invocation latency means that an application cannot meet specified timing requirements as it cannot control the timing of access to resources. This property holds for all of the current serverless computing offerings [2, 15, 23]. If the invocation latencies lag the application requirement, then real-time and quality requirements (SLO) will not be achieved.

Real-time Serverless has been proposed as a modest extension of the Serverless interface [27], adding an interface for applications to advertise a guaranteed invocation rate, and a service-level objective (SLO) to deliver that rate as follows. Each real-time serverless function is defined with a finite guarantee invocation rate A. Given this rate, the application is guaranteed that for any period of $1/A$ time units, they will obtain at least one serverless invocation of the function. For example, a function with a guaranteed invocation rate of 5 invocations per second ensures that the application will receive invocations at least once every 200 milliseconds. The guarantee ensures the minimum invocations delivered to the applications is a linear function of time

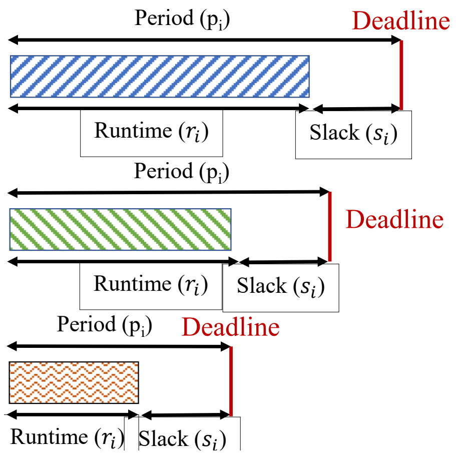


Figure 2: Rate Monotonic Application formulation: each task has a runtime, period/deadline, and slack.

whose slope is determined by the rate-guarantee as shown in Figure 1b (green line). If the guaranteed invocation rate is configured to match the load, then the application is ensured to get sufficient invocations on time which makes it possible to cap the computation time, and thus ensuring hard real-time deadlines. However, to date, no rigorous analysis shows that a guaranteed invocation rate is sufficient to support real-time guarantees, based on dynamic allocation.

2.2 Rate-monotonic Scheduling

Rate-monotonic scheduling is a well-studied formulation of a set of real-time tasks, and deadlines [7,19]. The application is a set of n tasks, each of which is periodic with period p_i, and runtime $r_i \leq p_i$. This situation is shown in Figure 2. In the simplest formulations, tasks are released at the beginning of each period. Early work proved that for a set of n tasks with unique periods, a feasible schedule that will meet all of the deadlines exists if CPU utilization is below the bound:

$$U = \sum_{i=1}^{n} \frac{r_i}{p_i} \leq n(2^{1/n} - 1) \tag{1}$$

Extensive work has explored bounds under varied assumptions and extensions of both workload, resource, and scheduling models [7]. However, this classic formulation frames a required CPU under-utilization of $(1-U)$ to achieve these real-time guarantees.

3. SERVERLESS AND REAL-TIME APPLICATIONS

The core benefit of the serverless compute model is dynamic allocation representing an opportunity to implement bursty, real-time systems without any wasted resources. This differs from most traditional real-time task scheduling models that depend on fixed/static allocation of compute resources. Furthermore, because the scaling is handled by the serverless systems, development and deployment efforts from the application end are minimized.

These advantages make implementing real-time applications using the serverless compute model an appealing approach. At the simplest implementation – executing each real-time task by a single serverless invocation – the application can effortlessly minimize resource consumption. However, due to the lack of insights into serving real-time tasks with a dynamically scaling compute model such as serverless, there is no rigorous way that shows us whether doing so can guarantee (or partly guarantee) the application's real-time deadlines. Even if the answer is yes, we still do not know how to properly configure and use serverless functions to meet real-time deadlines.

To fill this gap, we construct an analytical framework that bridges real-time deadlines with serverless serving by mapping the execution of rate-monotonic workload onto serverless dynamic allocation. Based on the mapping, we develop theory and mathematical proofs around task execution analysis to characterize serverless real-time support and get insights into how a guaranteed invocation rate could resolve the situation.

4. ANALYTICAL FRAMEWORK

Our approach exploits the dynamic allocation in the serverless model, and the guaranteed allocation rate of the real-time serverless model. Our goal is to support periodic real-time applications, such as those described in a rate-monotonic workload, but with dynamic allocation, so minimum compute resources are used.[2] Consider a *rate-monotonic* real-time workload [19] with n periodic real-time tasks $T_1, ..., T_n$ each characterized by

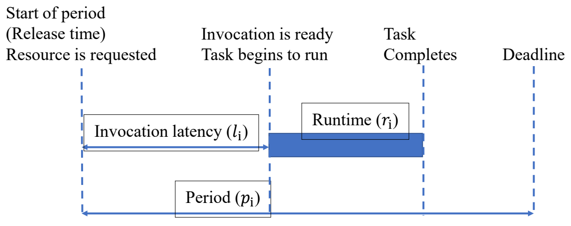
- Period (p_i): the task recurs every p_i time units. For simplicity, we assume tasks are released at the beginning of the period, and have to finish by the end of the period (i.e., the hard real-time deadline).
- Task runtime (r_i): the time for task to run
- Task slack $s_i = p_i - r_i$: the time a task does not spend on execution within a period.

We consider two ways of implementing a task T_i:

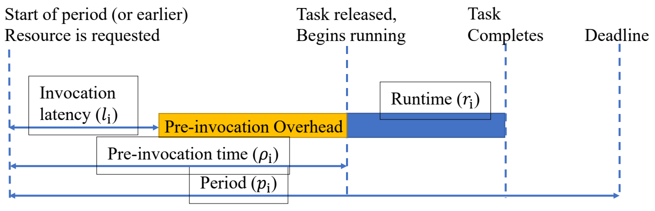
- **Serverless**: stateless functions invocations serve each task at one invocation per task release.
- **Real-time Serverless (RTS)**: similar to serverless, except that for each function, invocation rates can be guaranteed (A_i), ensuring that the number of invocations provided must be greater than 1 for any arbitrary period of length $\frac{1}{A_i}$.

Due to implementation overhead (e.g., initialization, resource allocation latency, etc.), both serverless and real-time serverless invocations have to wait for l_i time unit(s) (*invocation latency*) after being requested to start execution. With real-time serverless, the guaranteed invocation rate promises at least 1 invocation for any $1/A_i$ period, thus the invocation latency is bounded by $l_i \leq 1/A_i$ as long as $A_i \geq \frac{1}{p_i}$ (i.e., the rate of task release does not exceed the guaranteed invocation rate). Based on the guaranteed invocation rate definition, serverless is equivalent to real-time serverless with a guaranteed invocation rate of zero which means its invocation latency is unbounded.

[2]we can also handle bursty versions of rate monotonic workloads where tasks conform to the rate monotonic structure *when they occur*, but they often simply don't appear in their periods.



(a) Task on Serverless invocation



(b) Task on Serverless with pre-invocation

Figure 3: Supporting periodic tasks dynamically on Serverless compute model.

Symbol	Time Interval	Note
p_i	Period	$p_i \geq r_i$
r_i	Runtime	$r_i > 0$
s_i	Slack for task $i = p_i - r_i$	
l_i	Invocation Latency for task T_i	
ρ_i	Pre-invocation time for task T_i	

Table 1: Rate Monotonic Notation for each task T_i

Figure 3 shows two examples of timing diagrams for single periodic task execution. Normally, an invocation is requested right at the time a task is released (Figure 3a), so the response time for task T_i would be l_i+r_i. To workaround invocation latency, invocations can be requested *in advance* to make it available at the time a task is released for immediate execution. We call this technique *pre-invocation* and use ρ_i to denote pre-invocation time as shown in 3b. Note that pre-invocation shortens task response time at the cost of unused resources (waste) when invocation gets ready before a task release (orange bar in Figure 3b). We summarize the framework notations in Table 1.

5. LIMITS OF SERVERLESS

Based on the rate-monotonic workload model, we can easily prove that serverless alone is unable to guarantee that the periodic tasks will meet their deadlines as follows.

THEOREM 5.1. *Serverless cloud functions cannot guarantee that a single periodic task in a rate-monotonic workload will meet its deadline.*

PROOF. Given a periodic task T_i, with invocation latency l_i and runtime r_i, the time to complete the task can be written as

$$l_i + r_i$$

Which, for the task to meet its deadline must be

$$l_i + r_i \leq p_i \implies l_i \leq p_i - r_i$$

Let $l_i = \tau$ for a serverless invocation and $Prob(\tau > x)$ be the probability that τ is greater than x. Because the serverless invocations are best effort, $0 < \tau < \infty$, and for any given $p_i - r_i$, the

$$Prob(\tau > p_i - r_i) > 0$$

This means that

$$Prob(l_i + r_i > p_i) > 0$$

that is the chance that the task misses its deadline is greater than zero – its real-time performance is not guaranteed. □

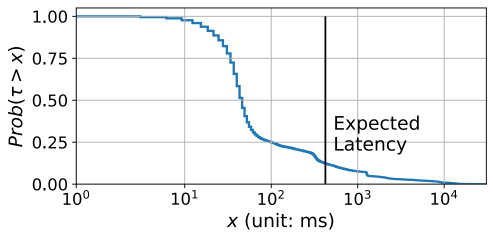


Figure 4: Google Cloud Functions invocation latency [35]

Figure 4 shows $Prob(\tau > x)$ estimated from invocation latency statistics of Google Cloud Functions [15,35], a commercial serverless platform. This is a long-tailed distribution. Invocation latency can exceed 30 seconds, more than 10-100x longer than the expected invocation latency [39]. This suggests that the unbounded invocation latency is a practical issue, not purely theoretical. Also, the latency is widely distributed leading to significant delays. This makes realizing real-time applications on top of best-effort serverless very challenging if not impossible.

Serverless is insufficient for even a single task, so we can easily show that it is unable to support the entire rate-monotonic workload of multiple periodic tasks.

THEOREM 5.2. *Serverless cloud functions cannot guarantee that a set of periodic tasks in a rate-monotonic workload will meet their deadlines.*

PROOF. Choose an arbitrary task T_i from the multiple periodic tasks. Theorem 5.1 shows that serverless cannot guarantee T_i will meet its deadline. Therefore, serverless cannot guarantee that all of the tasks in the rate monotonic workload meet their deadlines. □

From the proofs above, it is clear that the unbounded invocation latency is the main reason for serverless limitations on guaranteeing real-time deadlines. Real-time serverless resolve the issue by their guaranteed invocation rate extension as will be shown next.

6. REAL-TIME SERVERLESS ENABLES REAL-TIME PERFORMANCE

The guaranteed invocation rate allows real-time serverless functions to bound invocation latency. We first prove that this bound can guarantee tasks meet their deadlines as long as the tasks have non-zero slack. Later, we will show how this requirement can be relaxed by using pre-invocation.

THEOREM 6.1. *Real-time serverless can guarantee one periodic task T_i meets its deadline, if it has slack of $s_i = (p_i - r_i) > 0$.*

PROOF. For T_i to meet its deadline, we must have

$$l_i + r_i \le p_i$$

or

$$l_i \le (p_i - r_i)$$

Real-time serverless provides a guarantee of invocation rate, such that there is at least one invocation in every period of length $1/A_i$, where A_i is the guaranteed invocation rate chosen for real-time serverless. This means that as long as $A_i \ge \frac{1}{p_i}$, l_i can be bounded as follows:

$$l_i \le \frac{1}{A_i}$$

So to meet the deadline, we must ensure that

$$\frac{1}{A_i} \le (p_i - r_i)$$

Which we can assure for any slack $s_i = (p_i - r_i) > 0$, by picking a sufficiently large $A_i \ge \frac{1}{p_i - r_i} \ge \frac{1}{p_i}$ and gives

$$l_i \le \frac{1}{A_i} \le (p_i - r_i)$$

Which is true because $s_i > 0$. So the deadline is met. □

Now, let us generalize Theorem 6.1, considering a workload with multiple periodic tasks $T_1, ..., T_n$.

THEOREM 6.2. *Real-time serverless can guarantee a set of n periodic tasks $T_1, ..., T_n$ meet their deadlines, if each has slack of $(p_i - r_i) = s_i > 0$.*

PROOF. Consider a task T_i, because it is served by a dedicated function, by Theorem 6.1, the invocation rate of

$$A_i = \frac{1}{p_i - r_i}$$

is sufficient for T_i to guarantee meeting its deadline.

We assume each single task T_i uses a dedicated function with a finite guaranteed invocation rate A_i. So from the application point of view, there is no invocation contention between the tasks. So, we can repeat the argument for each of the other tasks, then the theorem is proved. □

7. EFFICIENT REAL-TIME GUARANTEE ON REAL-TIME SERVERLESS

Theorems presented in Section 6 prove real-time serverless' capability on ensuring real-time deadlines of rate monotonic applications. The one restriction was that the rate-monotonic tasks have non-zero slacks. Further, as in Theorem 6.1, we can see that for small slack, the required guaranteed invocation rate can be high. For example, if a task has a period of 15 seconds, and a slack of only 1 second, the resulting required guaranteed invocation rate would be 1/second, or 15 times the task rate. While in many realistic settings, many applications have non-zero, or better yet, a large slack for each task, the properties proved in Theorem 6.2 can be sufficient. However, to go further, in this section, we will relax this restriction, using pre-invocation, and further show that pre-invocation can dramatically reduce the rate requirements.

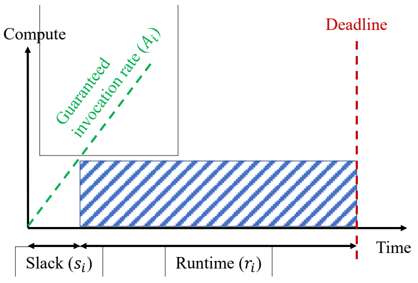


Figure 5: Tasks with short slack require high invocation rate guarantees.

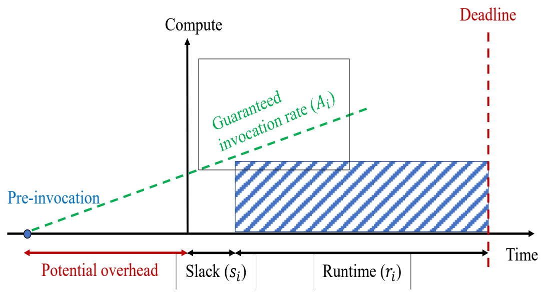


Figure 6: Pre-invocation reduces invocation rate guarantee requirement at a waste in resources.

The idea of pre-invocation arises from the notion that real-time serverless depends exclusively on the dynamic acquisition of resources from the underlying resource management system (as does serverless). This means the delay in acquiring such resources is critical in delivering real-time guarantees. That connection is illustrated in Figure 5 for a single rate monotonic task. The required guaranteed invocation rate is determined by the slack, and is much greater than $\frac{1}{p_i}$ – though intuitively that rate matches the average needs of the rate monotonic task.

Pre-invocation allocates resources early, anticipating the arrival of a task as shown in Figure 6. Because the resources are acquired before they are needed, it wastes resources. But as we will see pre-invocation can be used to significantly reduce the invocation rate guarantee requirement.

7.1 Efficient Real-time Guarantee for Single Task

First, we explore pre-invocation for a single task.

THEOREM 7.1. *With a finite pre-invocation of ρ_i, an application can use real-time serverless to guarantee deadlines of one periodic task T_i with any zero or positive slack (i.e., $s_i = p_i - r_i \ge 0$). This loosens the requirement of Theorem 6.1.*

PROOF. Assume the task T_i employs pre-invocation of

$$\rho_i = r_i$$

Then by serving T_i with real-time serverless of rate

$$A_i = \frac{1}{p_i}$$

Its invocation latency is bounded as

$$l_i \leq \frac{1}{A_i} = p_i$$

There are two possible cases

- $l_i \leq \rho_i$ meaning there is an invocation available at the time the task releases, so it requires only r_i to complete and

  $$r_i \leq p_i$$

  so the deadline is met.

- $\rho_i < l_i \leq p_i$ meaning the task has to wait for its invocation, so it waits $(l_i - \rho_i)$, and completes in

  $$(l_i - \rho_i) + r_i$$

  which is equal to l_i, and $l_i \leq p_i$, so the deadline is met.

Thus, pre-invocation with real-time serverless can guarantee a single periodic task meeting its deadline, even if the task has no slack. □

With Theorem 7.1, real-time serverless is sufficient for *any* single periodic task to guarantee its deadlines. Now, let us consider the cost of achieving real-time guarantees. The cost has two components: pre-invocation (wasted computation) and guaranteed invocation rate (higher rate-guarantee requires more implementation effort and hence more costly [27]). With our model, we study the interplay between these costs. First, given Theorem 7.1 and a finite, but very small pre-invocation, we will show that real-time guarantees can be met with essentially no pre-invocation overhead and a guaranteed invocation rate of $A_i \geq \frac{1}{p_i}$ as follows.

Theorem 7.2. *A single periodic task T_i requires at least an invocation rate*

$$A_i \geq \frac{1}{p_i}$$

to meet its deadline.

Proof. We will prove by contradiction. That is, assuming that T_i is guaranteed to meet its deadlines at invocation rate of $A'_i < \frac{1}{p_i}$. Now, let

$$\epsilon = \frac{1}{p_i} - A'_i > 0$$

Consider an interval of length $m = \frac{1}{\epsilon}$. Let I_i be the number of invocations needed to be completed by T_i within this interval, then

$$I_i \geq \lfloor \frac{m}{p_i} \rfloor$$

while the number of invocations we are guaranteed to have at the rate A'_i is

$$N_i = \lfloor m \cdot A'_i \rfloor$$

A necessary condition to guarantee that T_i does not miss any deadline over m is

$$N_i \geq I_i$$

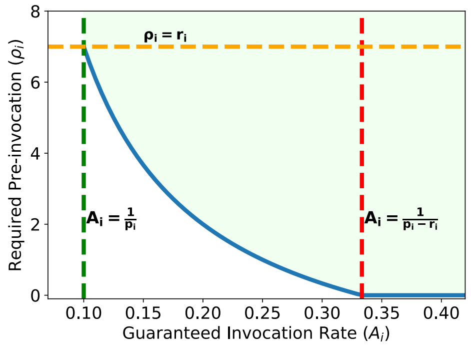


Figure 7: Required pre-invocation to guarantee real-time deadlines of a single task $T_i(p_i = 10, r_i = 7)$ varying guaranteed invocation rate A_i. The colored area shows (rate-guarantee, pre-invocation) combinations that ensure the task's real-time deadlines.

However, because

$$(\frac{m}{p_i}) - (m \cdot A'_i) = m(\frac{1}{p_i} - A'_i) = \frac{1}{\epsilon}\epsilon = 1$$

Then

$$I_i - N_i \geq \lfloor \frac{m}{p_i} \rfloor - \lfloor m \cdot A'_i \rfloor = 1$$

This means, $I_i > N_i$ so T_i is unable to guarantee its deadlines contradicting to the hypothesis. Thus, the invocation rate must be at least $\frac{1}{p_i}$. This proves the theorem. □

More generally, let's derive a precise expression for the required pre-invocation, given a sufficient guaranteed invocation rate:

Theorem 7.3. *Given task guaranteed invocation rate of $A_i \geq \frac{1}{p_i}$, we can ensure task T_i meeting its deadline with a pre-invocation time of*

$$\rho_i \geq \frac{1}{A_i} - (p_i - r_i)$$

Proof. Let us consider two possible cases

- *Case 1.* $\rho_i \geq \frac{1}{A_i}$, then

  $$l_i \leq \frac{1}{A_i} \leq \rho_i$$

  then there is always an available invocation before the task releases meaning it only requires r_i to complete and

  $$r_i \leq p_i$$

  so the deadline is met.

- *Case 2.* $\frac{1}{A_i} - (p_i - r_i) \leq \rho_i < \frac{1}{A_i}$ then l_i is no longer bounded by ρ_i. The invocation may arrive before the task releases then it falls back to Case 1, where T_i meets the deadline. Otherwise, T_i has to wait for invocation after releasing so it would take the task RT_i

time unit(s) to complete, where RT_i is determined as

$$\begin{aligned} RT_i &= l_i - \rho_i + r_i \\ &\leq \frac{1}{A_i} - \rho_i + r_i \\ &\leq \frac{1}{A_i} - [\frac{1}{A_i} - (p_i - r_i)] + r_i \\ &= p_i \end{aligned}$$

Thus, T_i also meets its deadline.

Therefore, $\rho_i \geq \frac{1}{A_i} - (p_i - r_i)$ ensures the task to guarantee meeting its deadline. □

Theorem 7.3 shows that given a sufficient guaranteed invocation rate, we can choose the minimum pre-invocation needed to enable a single task to ensure its real-time deadlines. This is the most efficient (least resource waste), given a sufficient guaranteed invocation rate.

The pre-invocation and guaranteed invocation rate relationship is shown in Figure 7. Realizing the lower invocation rate bound $A_i = \frac{1}{p_i}$ (green line) requires a pre-invocation of $p_i = r_i$ (orange line). As A_i increases, the invocation latency bound gets tighter, then the required pre-invocation decreases proportionally. Finally, at $A_i = \frac{1}{p_i - r_i}$ (red line), the slack is large enough so no pre-invocation is needed.

7.2 Efficient Real-time Guarantee for Multiple Tasks

Now, let us generalize the results above to the case of multiple periodic tasks.

THEOREM 7.4. *Pre-invocation enables real-time serverless to guarantee* $\underline{\text{many}}$ *periodic tasks $T_1, ..., T_n$ meeting their deadlines without the positive slack requirement (i.e., $s_i = p_i - r_i \geq 0$).*

PROOF. Given a task T_i, let us request an invocation for each of its releases in ρ_i time unit ahead, where

$$\rho_i = r_i$$

Now, by Theorem 7.3, this pre-invocation enables T_i to achieve its real-time guarantee with a finite invocation rate

$$A_i = \frac{1}{p_i}$$

Applying the same argument for other tasks then all the tasks are guaranteed to meet their deadlines. This proved the theorem. □

Similar to Theorem 7.1, Theorem 7.4 extends Theorem 6.2's scope to tasks with no slack which finally, proves that real-time serverless is sufficient for *any* combination of multiple periodic tasks to achieve real-time guarantee.

Now, we will generalize theorems on the bound of the cost to multiple tasks. However, representing the cost by a set of guaranteed invocation rates used by each task is complicated, hard to analyze, and make comparisons. Hence, we consider the invocation cost as the invocation rate for the application as a whole. In particular, given multiple tasks $T_1, ..., T_n$ served with invocation rates of $A_1, ..., A_n$ then the invocation cost for these tasks would be the sum of the number of invocations created at rates $A_1, ..., A_n$ per time unit.

THEOREM 7.5. *A real-time serverless system can meet the deadlines for rate-monotonic workload with periodic tasks $T_1, ..., T_n$ given a guaranteed invocation defined as*

$$A_{total} \geq \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \frac{1}{p_i}$$

PROOF. Consider the time interval of length M:

$$M = \prod_{j=1}^{n} p_j$$

Clearly, M is a common multiple of $p_1, ..., p_n$ so if we are able to ensure $T_1, ..., T_n$ to meet their deadline within this interval, they are guaranteed to meet deadlines in any interval. Consider a task T_i, let N_i be the number of its releases within the interval, then

$$N_i = \frac{M}{p_i} = \frac{\prod_{j=1}^{n} p_j}{p_i} = \prod_{j \neq i} p_j$$

Thus, the total number of task releases is

$$N_{total} = \sum_{i=1}^{n} N_i = \sum_{i=1}^{n} \prod_{j \neq i} p_j$$

Clearly, there would be N_{total} invocation requests within the interval so in order to ensure that no task misses its deadline, the invocation rate must be fast enough to make at least N_{total} invocations available. Therefore, the lower bound for the shared guaranteed invocation rate is

$$A_{total} \geq \frac{N_{total}}{M} = \sum_{i=1}^{n} \frac{\prod_{j \neq i} p_j}{M} = \sum_{i=1}^{n} \frac{1}{p_i} = \sum_{i=1}^{n} A_i$$

Thus, the theorem is proved. □

Note that $A_{total} = \sum_{i=1}^{n} A_i$ is just a lower bound for invocation rate, stating how *fast* the serverless system should deliver their invocations, not *when* should they deliver invocations. In fact, given A_{total}, the serverless system can even decompose it back to $A_1, ..., A_n$ where $\sum_{i=1}^{n} A_i = A_{total}$ to serve $T_1, ..., T_n$ individually.

THEOREM 7.6. *Invocations delivered at rate A_{total} can be partitioned to form n different invocation rates $A_1, ..., A_n$ where*

$$A_{total} = \sum_{i=1}^{n} A_i$$

PROOF. Consider an arbitrary interval of length T, the number of invocations guaranteed to be available within T at rate A_i is

$$N_i = \lfloor T \cdot A_i \rfloor$$

while the number of invocations delivered by A_{total} is

$$N_{total} = \lfloor T \cdot A_{total} \rfloor = \lfloor T \cdot \sum_{i=1}^{n} A_i \rfloor \geq \sum_{i=1}^{n} \lfloor T \cdot A_i \rfloor = \sum_{i=1}^{n} N_i$$

Thus, at any interval, invocations given at rate A_{total} is always greater than or equal to the total number of invocations needed by $A_i, ..., A_n$. Therefore, by temporally shifting invocations created at rate A_{total} within the interval, we can form n guaranteed invocations. This proves the theorem. □

The theorem states that different real-time deadlines can be ensured *simultaneously* with a single guaranteed invocation rate. Combined with Theorem 7.5, we provide important results that allow us to ensure hard real-time deadlines with great flexibility that can be essential to deal with different practical scenarios. For example, we can multiplex different tasks into a single real-time serverless function to simplify function management given a large number of periodic tasks. Or if the rate-guarantee of a function is too high making its deployment impracticable, we can decompose it into identical ones with smaller rates. Finally, we use these theorems to realize the lower pre-invocation bound for A_{total}.

THEOREM 7.7. *The lower bound from Theorem 7.5 for guaranteed invocation rate of*

$$A_{total} = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \frac{1}{p_i}$$

can be achieved with total pre-invocation overhead (wasted compute) of

$$\rho_{total} = \sum_{i=1}^{n} \rho_i = \sum_{i=1}^{n} r_i$$

PROOF. By applying Theorem 7.6, we can decompose A_{total} into

$$A_i = \frac{1}{p_i}$$

By using A_i to serve T_i, Theorem 7.3 indicates that a pre-invocation of

$$\rho_i = r_i$$

is needed to achieve its real-time guarantee. Applying the argument for other tasks, then at the invocation rate of

$$A_{total} = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \frac{1}{p_i}$$

the real-time guarantees are met only with total pre-invocation of

$$\rho = \sum_{i=1}^{n} \rho_i = \sum_{i=1}^{n} r_i$$

This makes the pre-invocation overhead (wasted compute) bounded by $\sum_{i=1}^{n} r_i$. Thus, the theorem is proven. □

Theorem 7.7 generalizes the conclusions of Theorem 7.3, showing that pre-invocation required to ensure no task misses its deadlines will never exceed the real computation cost. In other words, a pre-invocation overhead of 100% is sufficient to reduce the guaranteed invocation rate requirements to their minimum, $A_{total} = \sum_{i=1}^{N} \frac{1}{p_i}$.

8. DEMONSTRATION

In this section, we will demonstrate the implications of the above theoretical results by using real-time serverless to serve a distributed real-time virtual/augmented reality (VR/AR) application. We aim to demonstrate how the application design and deployment can be enhanced by utilizing our theoretical findings, leading to improved user experience, simplified deployment, and streamlined management.

Task	Period	Runtime	Slack
Stream (T_1)	$p_1 = 30$	$r_1 = 15$	$s_1 = 15$ (50%)
Handle (T_2)	$p_2 = 50$	$r_2 = 25$	$s_2 = 25$ (50%)
Sync (T_3)	$p_3 = 90$	$r_3 = 50$	$s_3 = 40$ (44%)

Table 2: Rate monotonic tasks collected from VR/AR applications (milliseconds).

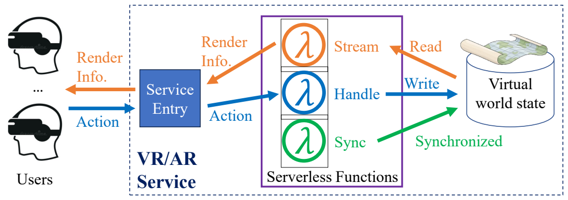


Figure 8: The VR/AR service. Requests come from various users creating corresponding serverless function invocations.

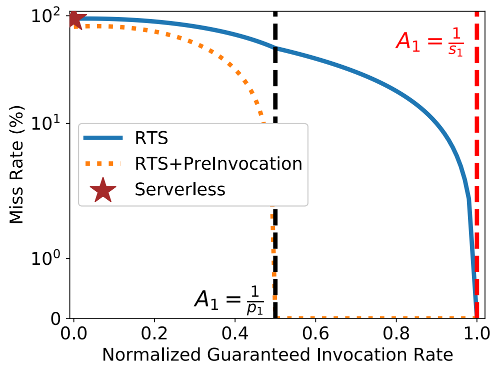
8.1 Setup

8.1.1 VR/AR Application and Workload

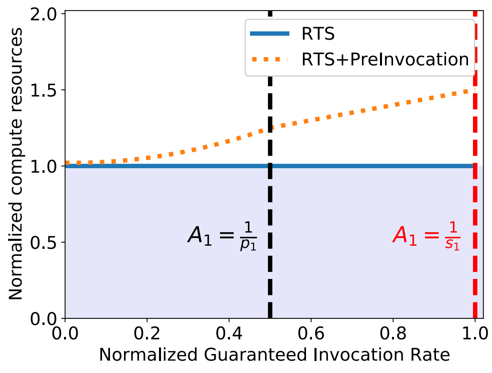
Distributed VR/AR has gained significant attention in recent years due to the innovative user experience it offers. Quality of experience (QoE), measured by the smoothness of user interaction, is one of its critical requirements. Maintaining high QoE requires timely processing in many tasks such as rendering, synchronizing multi-player actions, etc. We will show that by specifying these tasks as hard real-time and enforcing their real-time deadlines using real-time serverless, we can unlock new, game-changing capabilities to these applications. These capabilities not only allow them to control QoE but also simplify their management in ever-changing workloads and deployment environments.

Motivated by [28], we model a VR/AR application as a cloud service that creates a virtual world serving as a common place for hundreds or even thousands of users to interact with each other simultaneously (Figure 8). The virtual world state, including users' locations, appearances, and movements, etc., is maintained in global storage and is continuously updated to match users' actions (e.g., talk, move, make a purchase, etc.). We select three representative time-sensitive tasks that are frequently executed by VR/AR applications and model them as a rate-monotonic workload with parameters listed in Table 2:

- *Stream*: reads the virtual world state, generates render information, and then sends it to the user's end devices for constructing the world from their point of view. With 30 fps is a standard for video streaming, we set the task period to 30ms and 15ms execution time.
- *Handle*: triggers when a user takes actions that require an immediate reaction from the application (e.g., talk, pick up an item, etc.). Based on in-game analysis [25], there can be up to 17 clicks per second in aggressive gaming situations, so we set the task period to 50ms with 25ms runtime.
- *Sync*: Synchronizes the virtual world's state, ensuring its consistency across users. We assume the application synchronizes the state once for every 3 video frames, so the task period is 90ms.

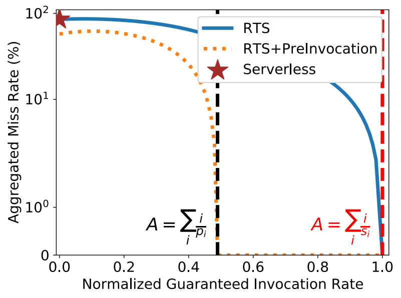


(a) Miss rate vs rate-guarantee (A_1, normalized to $\frac{1}{s_1}$). Pre-invocation: $\rho_1 = p_1 - s_1$

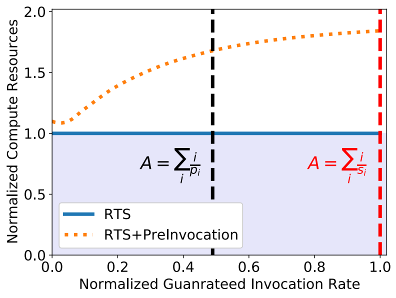


(b) Compute resources normalized to useful ones (colored area) vs. rate-guarantee (A_1, normalized to $\frac{1}{s_1}$)

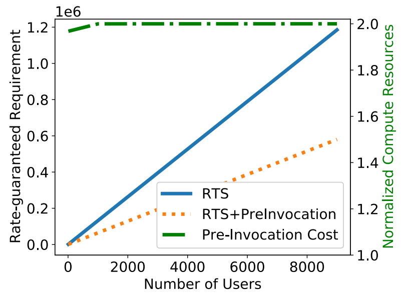
Figure 9: Implementing event streaming (i.e., T_1 – *Stream*) using Real-time Serverless ensuring the task's real-time deadlines at bounded resource cost.



(a) Miss rate vs rate-guarantee (A, normalized to $\sum_i \frac{1}{s_i}$). Pre-invocation: $\rho_i = p_i - s_i$



(b) Compute resources normalized to useful computation (colored area) vs. Rate-guaranteed (A) normalized to $\sum_i \frac{1}{s_i}$



(c) Rate-guarantee and resources requirement (normalized to useful computation) varying number of active users.

Figure 10: Serving combinations of multi-tasks with Real-time Serverless. Real-time Serverless ensures real-time guarantee for any task combination with a finite rate-guarantee at a resource cost bounded by 2x of useful computation.

8.1.2 Approaches

We compare these task executions on regular serverless, real-time serverless (RTS), and real-time serverless with pre-invocation (RTS+PreInvocation) via simulation. Every time a task releases, it needs an invocation for execution. In the base case, the task requests a new invocation at the beginning of its period. If pre-invocation is enabled, then the task makes invocation requests earlier than the beginning of its periods. If the task requests a regular serverless invocation, the invocation latency is simulated based on Google Cloud Functions cold start latency statistics [35] (Figure 4). For real-time serverless invocations, we collect the invocation latency statistics from deploying and executing functions with equivalent rate-guarantees on a real-time serverless prototype [27]. Once a task gets an invocation, it starts the execution with a constant runtime (r_i, Table 2). If an execution completes *after* the beginning of the next period, we count it as *missing the deadline*. We report the percentage of executions missing the deadlines (i.e., *miss rate*) as a QoE metric. The *compute resource* (or resource usage), calculated by aggregating the invocation lifetime, including the pre-invocation overhead , is a cost metric.

8.2 Results

8.2.1 Single Task

In Figure 9a, we plot the percentage of missed deadlines for a single task "*Stream*" (T_1) implemented by different approaches mentioned above. The regular serverless leaves invocation latency unbounded so many invocations fail to start within the task's slack – 15ms leading to more than 85% miss rate (the red star at the top-left corner). Real-time serverless enables bounding the latency through the rate-guarantee A_1 so the higher the rate, the tighter the bound and thus, the lower the miss rate. Once $A_1 = \frac{1}{s_1} = 66.67$ invocations per second, the task is guaranteed to meet its deadlines, validating Theorem 6.1. With pre-invocation, the task uses pre-invocation of $\rho_1 = r_1 = 15ms$ ahead, equal to the upper bound for efficiency. Doing so adds extra time waiting for the new invocation, so T_1 sees lower deadline misses than real-time serverless alone (the orange line vs. the blue line). Pre-invocation eliminates the deadline misses at a much lower guaranteed invocation rate, $33.33 = 1/p_1$ invocations per second, confirming Theorem 7.2. By pre-invocation, however, applications have to hold invocations longer than usual if they arrive before the task released. This increases resource usage as shown in Figure 9b. Further, as the invocation rate increases, invocation arrives faster and thus, creates more overhead yet it never exceeds 100% of the total runtime, as shown in Theorem 7.3.

8.2.2 Multiple Tasks

Next, we consider deploying all tasks in Table 2 simultaneously. The theorem 7.6 indicates that we do not need to deploy each task with a separate real-time serverless function. Instead, only one real-time serverless function with a rate-guarantee equal to the total per-task rate requirement is sufficient. Figure 10a confirms this implication as at a guaranteed invocation rate of $A = \sum_i \frac{1}{s_i} \approx 132$ invocation/sec, a single real-time serverless function reduces all three tasks' aggregated miss rate to zero (blue curve) – ensuring all real-time deadlines are met. Further, Theorem 7.7 implies that we can even ensure real-time deadlines with an even lower rate-guarantee through pre-invocation. Every time a task T_i releases, we pre-invoke a new invocation r_i seconds ahead. As a result, resources become available for the task sooner, reducing the miss rate (the orange curve). And as proved in Theorem 7.7, $A = \sum_i \frac{1}{p_i} \approx 64$ invocations per second is already enough for guaranteeing real-time deadlines. Also similar to the case of a single task, pre-invocation incurs high resource use, but by Theorem 7.7, the overhead never exceeds 100% of useful resources (light blue area) as shown in Figure 10b.

8.2.3 Distributed Deployment

Finally, we consider the distributed deployment of the application where there can be thousands of users simultaneously interacting at a time. Yet the number of active users may vary widely, especially during special events, such as launch time, anniversary, etc., applications expected to obtain a burst load of 10x or more active users than the average [28]. In traditional deployments, this requires careful preparation to make just enough room for the burst to ensure the desired QoE at a reasonable cost.

With real-time serverless, the solution is much simpler. All the application has to do is simply reconfigure the serverless guaranteed invocation rate to match the task release rate at burst, as demonstrated in the previous experiments. Figure 10c shows the guaranteed invocation rate required for a single real-time serverless function to meet the real-time deadlines of tasks released by different numbers of users, ranging from 0 to more than 8,000. Since guaranteed invocation rates are combinable, the required rate-guarantee increases linearly with the number of active users demonstrating good scalability. Further, with pre-invocation, we can reduce the rate by half at the additional resource uses of at most 100% of the available resources.

It's worth noting that the rate-guarantee is not only combinable but can also be decomposed into smaller ones if needed. For instance, to serve 8,000 users, the required rate-guarantee is over 1 million invocations per second, which may exceed the current capability of serverless technologies. The application can decompose the serverless function into others with lower rate-guarantees and distribute them across different cloud regions (e.g., 1000 functions with 1000 invocation/sec, each serving users from a specific area across the globe). This approach not only works around current technology limitations but also leverages distributed resources deployment (e.g., cloud+edge) to achieve better load balancing and cost efficiency.

9. DISCUSSION AND RELATED WORK

To the best of our knowledge, there is currently no compute model offering resources with performance guarantee at the cost scale to actual demand. Therefore, real-time systems must rely on Virtual Machines or dedicated hosts [5] with indefinite runtime agreement and full resource management control to deploy their real-time guarantee solutions. Due to the long pricing period (hours) and high allocation latency, real-time systems that pursue this approach often have to trade off resource waste for performance guarantee [12, 19]. Still, many efforts trying to improve dynamic scaling unbounded latency such as AWS provisioned concurrency [4], Serverless, and many dynamic allocation frameworks (e.g., [17, 18, 20, 21, 42, 47]) but these works are more about attempting fast react to demand changes rather than performance guarantee.

There is a host of real-time scheduling studies that builds on rate-monotonic scheduling or other approaches to achieve real-time guarantees [12], mostly using fixed, stable performance resources [7,19]. Although there are attempts seeking for understanding of how real-time deadlines will be affected by dynamic execution [40] and its solutions (e.g., [22, 31]), to our knowledge, no work that focuses on dynamically allocated such as serverless, and seeks to achieve hard real-time performance with no resource waste or with bounded resource waste.

Utilizing serverless compute model for real-time applications is attractive, but challenging due to performance limitations and implementation issues, such as network latency, data access, etc. [10,45] Many studies focus on these issues, including virtualization and scheduling for more reliable execution [1,6,11,43,44], real-time data access [24,32,34], and deterministic networking [26,46].

10. SUMMARY AND FUTURE WORK

We have analyzed a proposed extension called *real-time serverless* that adds a guaranteed invocation rate to the serverless compute model with the goal of supporting hard real-time applications. Using a rate-monotonic workload example, our analysis shows that this new model can support hard real-time guarantees, but if the slack for tasks is short, the required invocation rates can be very high. However, because the serverless models support dynamic allocation, this approach meets real-time deadlines with no resource waste. Next, we showed that pre-invocation can reduce these required invocation guarantees; limiting them to $\frac{1}{p_i}$ for each task, at an overhead in compute resource waste. Together, these results provide a foundation for hard real-time applications on the real-time serverless platform.

There are a number of interesting directions for future work, including studies of how to schedule rate-monotonic tasks with dynamic runtimes for serverless systems as well as how to best implement the real-time serverless guaranteed invocation rates given its current implementation is too costly for bursty workloads [27]. Another interesting direction is to find the best way to utilize the guaranteed invocation rate to not only ensure applications' real-time deadlines but also to enable new capabilities for better application design and scalability.

Acknowledgment

We thank the anonymous reviewers for their insightful reviews, including those who reviewed the earlier versions of this paper. This work is supported in part by NSF Grants CMMI-1832230, OAC-2019506, CNS-1901466, and the VMware University Research Fund. We also thank the Large-scale Sustainable Systems Group members for their support of this work!

11. REFERENCES

[1] L. Abeni, A. Balsini, and T. Cucinotta. Container-based real-time scheduling in the linux kernel. *ACM SIGBED Review*, 16(3):33–38, 2019.

[2] AWS Lambda. `https://aws.amazon.com/lambda/`, 2015.

[3] L. Ao and et. al. Sprocket: A Serverless Video Processing Framework. In *SoCC '18*, 2018.

[4] Provisioned Concurrency for Lambda Functions. `https://aws.amazon.com/blogs/aws/new-provisioned-concurrency-for-lambda-functions/`.

[5] M. Azure". Azure dedicated host. `https://azure.microsoft.com/is-is/services/virtual-machines/dedicated-host/`.

[6] M. Barletta, M. Cinque, L. De Simone, and R. Della Corte. Achieving isolation in mixed-criticality industrial edge systems with real-time containers. In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[7] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Springer Publishing Company, Incorporated, 3rd edition, 2011.

[8] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard. Funcx: A federated function serving fabric for science. In *Proceedings of the 29th International symposium on high-performance parallel and distributed computing*, pages 65–76, 2020.

[9] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada. Fog function: Serverless fog computing for data intensive iot services. In *2019 IEEE International Conference on Services Computing (SCC)*, pages 28–35. IEEE, 2019.

[10] M. Cinque. Real-time faas: serverless computing for industry 4.0. *Service Oriented Computing and Applications*, 17(2):73–75, 2023.

[11] M. Cinque, R. Della Corte, A. Eliso, and A. Pecchia. Rt-cases: Container-based virtualization for temporally separated mixed-criticality task sets. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[12] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):1–44, 2011.

[13] Function-as-a-Service Market. `https://www.reportsanddata.com/report-detail/function-as-a-service-faas-market`.

[14] S. Fouladi and et. al. Encoding, Fast and Slow: Low-latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX NSDI'17*, 2017.

[15] Google Cloud Functions. `https://cloud.google.com/functions/`.

[16] M. Großmann, C. Ioannidis, and D. T. Le. Applicability of serverless computing in fog computing environments for iot scenarios. In *Proceedings of the 12th IEEE/ACM international conference on utility and cloud computing companion*, pages 29–34, 2019.

[17] A. Harlap, A. Chung, A. Tumanov, G. R. Ganger, and P. B. Gibbons. Tributary: spot-dancing for elastic services with latency slos. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 1–14, 2018.

[18] W. Lin, J. Z. Wang, C. Liang, and D. Qi. A threshold-based dynamic resource allocation scheme for cloud computing. *Procedia Engineering*, 23:695–703, 2011.

[19] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[20] F. Liu, K. Keahey, P. Riteau, and J. Weissman. Dynamically Negotiating Capacity between On-demand and Batch Clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 38. IEEE Press, 2018.

[21] X. Ma, H. Xu, H. Gao, and M. Bian. Real-time multiple-workflow scheduling in cloud environments. *IEEE Transactions on Network and Service Management*, 18(4):4002–4018, 2021.

[22] M. Masmano, I. Ripoll, A. Crespo, and J. Real. Tlsf: a new dynamic memory allocator for real-time systems. In *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.*, pages 79–88, 2004.

[23] Microsoft Azure Cloud. `https://azure.microsoft.com/`, 2010.

[24] G. Mondragón-Ruiz, A. Tenorio-Trigoso, M. Castillo-Cara, B. Caminero, and C. Carrión. An experimental study of fog and cloud computing in cep-based real-time iot applications. *Journal of Cloud Computing*, 10(1):32, 2021.

[25] Most actions per minute in a videogame. `https://www.guinnessworldrecords.com/world-records/88069-most-actions-per-minute-in-a-videogame`.

[26] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research. *IEEE Communications Surveys & Tutorials*, 21(1):88–145, 2018.

[27] H. Nguyen, C. Zhang, Z. Xiao, and A. A. Chien. Real-time serverless: Enabling application performance guarantees. In *Workshop on Serverless Computing*, December 2019.

[28] H. D. Nguyen, Z. Yang, and A. A. Chien. Motivating high performance serverless workloads. In *Proceedings of the 1st Workshop on High Performance Serverless Computing*, pages 25–32, 2020.

[29] OpenFaaS. `https://docs.openfaas.com`.

[30] A. Passwater. 2018 Serverless Community Survey: Huge Growth in Serverless Usage.

https://serverless.com/blog/2018-serverless-community-survey-huge-growth-usage Visited May, 2019.

[31] Q. Peng, H. Wu, and R. Xue. Review of dynamic task allocation methods for uav swarms oriented to ground targets. *Complex System Modeling and Simulation*, 1(3):163–175, 2021.

[32] A. Poniszewska-Maranda, R. Matusiak, N. Kryvinska, and A.-U.-H. Yasar. A real-time service system in the cloud. *Journal of Ambient Intelligence and Humanized Computing*, 11:961–977, 2020.

[33] S. R. Poojara, C. K. Dehury, P. Jakovits, and S. N. Srirama. Serverless data pipeline approaches for iot data in fog and cloud computing. *Future Generation Computer Systems*, 130:91–105, 2022.

[34] K. Ramamritham, S. H. Son, and L. C. DiPippo. Real-time databases and data services. *Real-time systems*, 28:179–215, 2004.

[35] Realtime serverless cell monitor. https://observablehq.com/@tomlarkworthy/serverless-cell-latency-monitor.

[36] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari. Mashup: making serverless computing useful for hpc workflows via hybrid execution. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 46–60, 2022.

[37] S. Sarkar, R. Wankar, S. N. Srirama, and N. K. Suryadevara. Serverless management of sensing systems for fog computing framework. *IEEE Sensors Journal*, 20(3):1564–1572, 2019.

[38] A. W. Services. Build a serverless real-time data processing app. https://amzn.to/2JcoFPp.

[39] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 205–218. USENIX Association, July 2020.

[40] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. *ACM Computing Surveys (CSUR)*, 50(2):1–40, 2017.

[41] J. Spillner, C. Mateos, and D. A. Monge. Faaster, better, cheaper: The prospect of serverless scientific computing and hpc. In *High Performance Computing: 4th Latin American Conference, CARLA 2017, Buenos Aires, Argentina, and Colonia del Sacramento, Uruguay, September 20-22, 2017, Revised Selected Papers 4*, pages 154–168. Springer, 2018.

[42] G. L. Stavrinides and H. D. Karatza. A hybrid approach to scheduling real-time iot workflows in fog and cloud environments. *Multimedia Tools and Applications*, 78:24639–24655, 2019.

[43] V. Struhár, M. Behnam, M. Ashjaei, and A. V. Papadopoulos. Real-time containers: A survey. In *2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[44] M. Szalay, P. Mátray, and L. Toka. Real-time task scheduling in a faas cloud. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 497–507, 2021.

[45] M. Szalay, P. Mátray, and L. Toka. Real-time faas: Towards a latency bounded serverless cloud. *IEEE Transactions on Cloud Computing*, 11(2):1636–1650, 2023.

[46] X. Yang, D. Scholz, and M. Helm. Deterministic networking (detnet) vs time sensitive networking (tsn). *Network*, 79, 2019.

[47] J. Zhou, J. Sun, M. Zhang, and Y. Ma. Dependable scheduling for real-time workflows on cyber–physical cloud systems. *IEEE Transactions on Industrial Informatics*, 17(11):7820–7829, 2020.