System Design: The Distributed Task Scheduler

Learn about the basics of designing a distributed task scheduler.

We'll cover the following

What is a task scheduler?#

A **task** is a piece of computational work that requires resources (CPU time, memory, storage, network bandwidth, and so on) for some specified time. For example, uploading a photo or a video on Facebook or Instagram consists of the following background tasks:

1. Encode the photo or video in multiple resolutions.
2. Validate the photo or video to check for content monetization copyrights, and many more.

The successful execution of all the above tasks makes the photo or video visible. However, a photo and video uploader does not need to stop the above tasks to complete.

Another example is when we post a comment on Facebook. We don't hold the comment poster until that comment is delivered to all the followers. That delivery is delegated to an asynchronous task scheduler to do offline.

In a system, many tasks contend for limited computational resources. A system that mediates between tasks and resources by intelligently allocating resources to tasks so that task-level and system-level goals are met is called a **task scheduler**.

When to use a task schedular#

A task scheduler is a critical component of a system for getting work done efficiently. It allows us to complete a large number of tasks using limited resources. It also aids in fully utilizing the system's resources, provides users with an uninterrupted execution experience, and so on. The following are some of the use cases of task scheduling:

- **Single-OS-based node:** It has many processes or tasks that contend for the node's limited computational resources. So, we could use a local OS task scheduler that efficiently allocates resources to the tasks. It uses multi-feedback queues to pick some tasks and runs them on some processor.

- **Cloud computing services:** Where there are many distributed resources and various tasks from multiple tenants, there is a strong need for a task scheduler to utilize cloud computing resources efficiently and meet tenants' demands. A local OS task scheduler isn't sufficient for this purpose because the tasks are in the billions, the

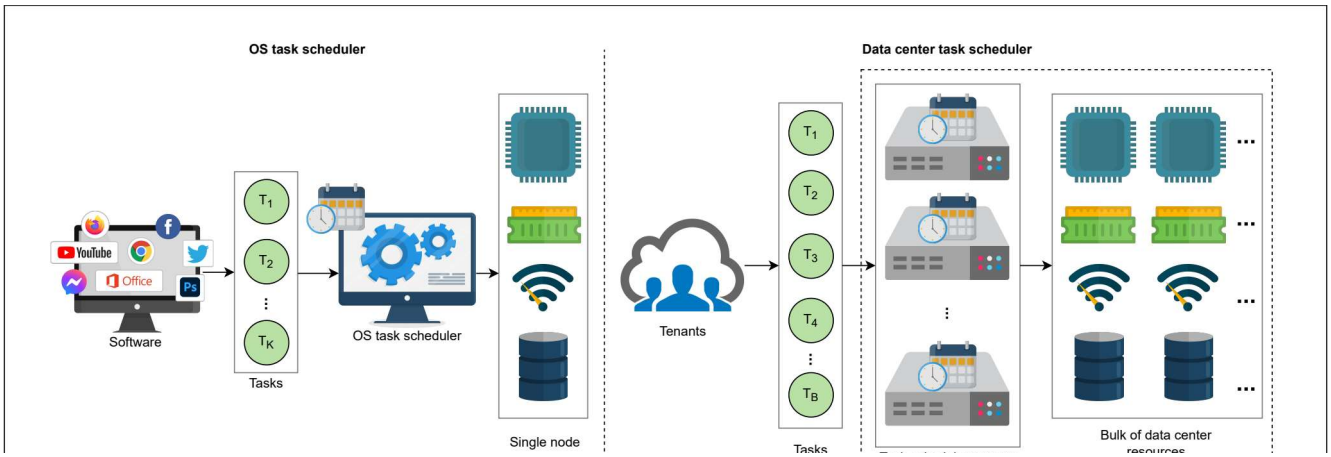
source of the tasks is not single, and the resources to manage are not in a single machine. We have to go for a distributed solution.

- **Large distributed systems:** In this system, many tasks run in the background against a single request by a user. Consider that there are millions to billions of users of a popular system like Facebook, WhatsApp, or Instagram. These systems require a task scheduler to handle billions of tasks. Facebook schedules its tasks against billions of parallel asynchronous requests by its users using Async.

> **Note:** Async is Facebook's own distributed task scheduler that schedules all its tasks. Some tasks are more time-sensitive, like the tasks that should run to notify the users that the livestream of an event has started. It would be pointless if the users received a notification about the livestream after it had finished. Some tasks can be delayed, like tasks that make friend suggestions to users. Async schedules tasks based on appropriate priorities.

Distributed task scheduling#

The process of deciding and assigning resources to the tasks in a timely manner is called **task scheduling**. The visual difference between an OS-level task scheduler and a data center-level task scheduler is shown in the following illustration:



An OS-level task scheduler vs. a data center-level task scheduler

The OS task scheduler schedules a node's local tasks or processes on that node's computational resources. At the same time, the data center's task scheduler schedules billions of tasks coming from multiple tenants that use the data center's resources.

Our goal is to design a task scheduler similar to the data center-level task scheduler where the following is considered:

- Tasks will come from many different sources, tenants, and sub-systems.
- Many resources will be dispersed in a data center (or maybe across many data centers).

The above two requirements make the task scheduling problem challenging. We'll design a distributed task scheduler that can handle all these tasks by making it scalable, reliable, and fault-tolerant.

How will we design a task scheduling system?#

We have divided the design of the task scheduler into four lessons:

1. **Requirements:** We'll identify the functional and non-functional requirements of a task scheduling system in this lesson.
2. **Design:** This lesson will discuss the system design of our task scheduling system and explores the components of the system and database schema.
3. **Design considerations:** In this lesson, we'll highlight some design factors, such as task prioritization, resource optimization, and so on.
4. **Evaluation:** We'll evaluate our design of task scheduler based on our requirements.

Let's start by understanding the requirements of a task scheduling system.

Back

System Design: Distributed Logging

Next

System Design: The Sharded Counters

Login to save progress

Report an Issue