Back To Module Home

Basic Building Blocks for Modern System Design

0% completed

Search Module

Introduction to Building Blocks

Domain Name System

Load balancers

Cache

Databases

Key-value Store

Content Delivery Network (CDN)

Sequencer

Distributed Monitoring

Distributed Cache

☐

Distributed Messaging Queue

☐

Pub-sub

☐

Rate Limiter

☐

Blob Store

☐

Distributed Search

☐

Distributed Task Scheduler

☐

Evaluation of a Distributed Task Scheduler's Design

Evaluate the proposed task scheduler system based on our requirements.

We'll cover the following

Availability#

The first component in our design was a rate limiter that is appropriately replicated and ensures availability. Task submission is done by several nodes. If a node that submits a task fails, the other nodes take its place. The queue in which we push the task is also distributed in nature, ensuring availability. We always have resources available because we continuously monitor if we need to add or remove resources. Each component in the

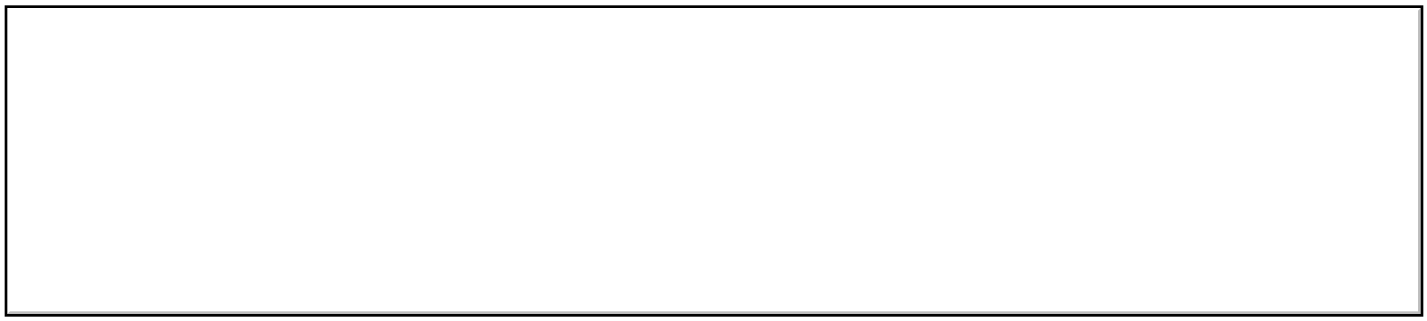
design is distributed and makes the overall system available.

Durability#

We store the tasks in a persistent distributed database and push the tasks into the queue near their execution time. Once a task is submitted, it is in the database until its execution.

Scalability#

Our task scheduler provides scalability because the task submitter is distributed in our design. We can add more nodes to the cluster to submit an increasing number of tasks. The tasks are then saved into a distributed relational database, which is also scalable. The tasks from RDB are then pushed to a distributed queue, which can scale with an increasing number of tasks. We can add more queues for different types of tasks. We can also add more resources depending on the resource-to-demand ratio.



Non-functional requirements fulfilled by our task scheduler system

Fault tolerance#

A task is not removed from the queue the first time it is sent for execution. If the execution fails, we retry for the maximum number of allowed attempts. If the task contains an infinite loop, we kill the task after some specified time and notify the user.

Bounded waiting time#

We don’t let the users wait for an infinite time. We have a limit on the maximum waiting time. If the limit is reached and we are unable to schedule the task for some reason, we notify the users and ask them to try again.

Conclusion#

We discussed the difference between an OS-level task scheduler and a data center-level task scheduler. We explained that the data center-level task scheduling needs a distributed solution because of multiple tenants and dispersed resources. We learned that the queue is a major building block of a task scheduler. We also used distributed queues where we can scale with an increasing number of tasks to utilize an increasing number of resources.

This lesson helped us to evaluate the issues with the FIFO queue. It was observed that the main job of the task scheduler is to set the priorities of the tasks for which we used a delay tolerance parameter. We discussed how the task scheduler determines the delay tolerance value and used different distributed databases to store task details. We ensured that the dependent tasks are executed in order by running the tasks according to DAG stored in the graph database. Depending upon the number of tasks (or demand), we added or removed resources to optimize the capacity. In the end, we used the monitoring service that alerts the administrators in case we need to add or remove resources.

Design Considerations of a Distribute…

Next

System Design: The Sharded Counters

Mark as Completed

Report an Issue