1. Is it ok to use a single http trigger to start multiple orchestrators in azure durable function?

It is important to note that starting multiple orchestrator instances concurrently can have performance and resource utilization implications. It is recommended to carefully consider the workload and concurrency requirements of your application before using this approach.

While calling multiple orchestrators is not an issue from one http trigger but orchestrators sharing the same function computer (app service plan) can be cause resource utilization implications as mentioned above.

2. Should we migrate all or some of user orchestrator to service bus triggers?

Migrating your orchestrator functions from HTTP triggers to Service Bus triggers can have some benefits, but it also depends on the specific requirements of your application. Here are a few factors to consider when deciding whether to migrate your orchestrator functions to Service Bus triggers:

1. Reliability: Service Bus triggers offer a higher level of reliability compared to HTTP triggers, as they can automatically retry failed messages and guarantee delivery of messages to the function.
2. Scalability: Service Bus triggers can scale more easily than HTTP triggers, as they can automatically scale up and down based on the volume of messages in the queue.
3. Asynchrony: Service Bus triggers allow you to decouple your orchestration logic from the caller, making it easier to build async, event-driven applications.

4. Cost: If you have a high volume of orchestration messages, using Service Bus triggers could potentially be more cost-effective compared to HTTP triggers.

Overall, whether to migrate your orchestrator functions to Service Bus triggers or not will depend on the specific requirements of your application. If you need a high level of reliability, scalability, and asynchrony, then Service Bus triggers might be a good choice. However, if your application has a lower volume of orchestration messages and does not require these features, then using HTTP triggers might be sufficient.

3. Are there specific tasks that are better suited to orchestrator vs service bus triggers?

Orchestrator functions and Service Bus triggers are both useful tools for building durable, scalable, and reliable applications in Azure Functions. However, they are best suited for different types of tasks.

Orchestrator functions are best suited for tasks that require long running, stateful logic and complex coordination across multiple functions. They allow you to build workflow-style applications that can execute logic over an extended period of time and manage the state of the workflow as it progresses.

Service Bus triggers, on the other hand, are best suited for tasks that require message processing, integration with other systems, and event-driven architectures. They allow you to build scalable, async applications that can process messages from a Service Bus queue or topic in a reliable and durable manner.

In general, if you need to build a workflow-style application that requires long-running, stateful logic and coordination across multiple functions, orchestrator functions might be the better choice. If you need to build a scalable, async application that processes messages from a Service Bus queue or topic, then Service Bus triggers might be the better choice.

4. Are we supposed to be using multiple threads in Orchestrator?

   Orchestrator functions in Azure Durable Functions are designed to be single threaded, meaning that they can only execute one task at a time. This is because orchestrator functions manage the state and progress of a long-running workflow and allowing multiple threads to execute concurrently could lead to race conditions and other issues.

   However, orchestrator functions do provide a way to parallelize work by using the Task.WhenAll method or the Parallel activity. These allow you to specify a set of tasks that should be executed concurrently and wait for all of them to complete before continuing.

   Using the Task.WhenAll method or the Parallel activity allows you to parallelize work within an orchestrator function, but it is important to note that each task is still executed on a single thread. If you need to scale out across multiple threads or machines, you can use other Azure Functions features such as Azure Functions scale controller or Azure Functions Scale Out with a Service Bus queue.

So, our recommendation is not to use durable function if multi-threading is need, please use service bus triggers for multi-threading/

Since might be the main reason your orchestration functions were failing in addition to lack of compute resources.

Supporting documents

https://microsoft-my.sharepoint.com/:v:/p/kevcarter/EeIzQy7hsrBNuKmwi2dxjWIBcDFbn1GuaAiPY196Jw4cZQ

5. Should we design orchestrators to limit runtime to the default value or less than 30 minutes?

Resource utilization: Running an orchestrator function for an extended period can potentially lead to increased resource utilization, such as CPU and memory usage. It is important to consider the workload and resource utilization of your orchestrator function and design it appropriately to avoid overloading the system.

In general, it is a good idea to design orchestrator functions to complete within the default execution timeout of 30 minutes, unless there is a specific reason to allow them to run for a longer period. This can help ensure that your orchestrator functions are efficient, cost-effective, and do not cause undue strain on the system.

‘

Other recommendations: Please make sure no deterministic coding (calling db. etc.) is done into orchestration function. All of this kind of job should be done in activities and orchestration function should be lightweight.

Durable orchestrator code constraints - Azure Functions | Microsoft Learn

Apart from all the above, as we discussed in the call, it is important that functions should be divided into different plans (Premium recommended because of its scaling ability).

Azure Functions Premium plan | Microsoft Learn