On-Demand Integration Modules: Milestone 1 Report

Date: 24.1.2025
Milestones completed: Milestone 1
Implementer/s: Szegoo (126X27SbhrV19mBFawys3ovkyBS87SGfYwtwa8J2FjHrtbmA)
Requested Payment: $18,000 USD to be converted based on Subscan EMA7
Short description: This report outlines the work completed in the first milestone of the on-demand integration modules. It is linked to the discussion that was accepted as part of this proposal: #243

---

1. Context of the proposal

This is definitely one of the most challenging things I have worked on so far. There were many times when I thought everything was good to go, only to realize the next day that there was a potential issue with the solution. This is why delivering this first milestone took a bit longer than I anticipated. However, I am happy with the progress made so far and believe the project is on a good track to being production-ready and used by new projects entering the ecosystem.

In this first milestone, we developed general on-demand integration modules intended for use by future on-demand parachains. There are several benefits to providing general on-demand modules, as implementing such logic is not a trivial task. Without highly customizable, ready-to-use modules, many parachain teams might ultimately avoid using on-demand functionality, even if it was their initial intention, due to the time-consuming development process and additional costs involved. Having common modules that are audited and well-documented will offer a significantly easier development path for on-demand projects on Polkadot.

In the first milestone, I delivered modules that support coordinated and automated order placement. A template is provided that uses the developed modules, allowing the chain to rely on the on-demand Coretime model.

2. Details of the report

Tasks	Status	Links to deliverables
Order placement service	Complete	The developed service automates order placement and is designed to be run by collator nodes within the network. It is highly configurable and generic, as it relies on a configuration type to specify all necessary details: https://github.com/RegionX-Labs/On-Demand/blob/master/services/order/src/config.rs To briefly explain, the service tracks new relay chain blocks to check if the parachain is registered as a parathread. If it is, and if there are cores assigned to the instantaneous Coretime pool, the service verifies whether the collator should create an order and proceeds accordingly.
On-demand pallet	Complete	The on-demand pallet stores configurations within the runtime and provides a way for them to be modified by an `AdminOrigin`. All the exposed extrinsics are described in the documentation: https://docs.regionx.tech/on-demand#pallet-on-demand-overview This pallet is also responsible for rewarding collators for order placement. The rewarding logic is not hardcoded in the pallet; instead, it expects an implementation to be provided within the pallet's `Config`. Currently, a default implementation is provided that mints a fixed amount of tokens to the order placer. The collator is rewarded only when they were expected to create an order; otherwise, no reward is given for order placement.
Template parachain	Complete	A template parachain is provided, utilizing the provided modules to function as a self-coordinated, automated on-demand parachain. Parachain: https://github.com/RegionX-Labs/On-Demand/tree/master/template A basic e2e test is also provided to ensure that the on-demand integration works. Link: https://github.com/RegionX-Labs/On-Demand/blob/master/template/e2e/order-placement.ts

Documentation	Complete	Documentation is available that provides a basic explanation of how the on-demand modules work and includes a step-by-step guide on integrating them with an existing parachain. It also explains all the configuration types and parameters, as well as the configuration extrinsics from the on-demand pallet. Link: https://docs.regionx.tech/on-demand
Credit purchase automation	Partially Complete / Change of Plans	The PR in which I implemented on-demand credits on the relay chain side is complete: https://github.com/paritytech/polkadot-sdk/pull/5990 The modules right now use the `place_order_allow_death` to place an order, and this will be updated to the new `place_order_with_credits` once there is a release with the new version of the pallet. I realized that providing credit purchase automation might not make much sense for now. This is something collators can handle manually from time to time. Still, it will be useful to provide a warning when collators are running low on credits.

3. Success metrics and feedback

The initial version of on-demand integration modules has been delivered.

4. Next steps

Ancestry Proof

In its current form, the logic for checking whether the collator is eligible for a reward does not verify ancestry proofs. Currently, the collator provides the relay block hash in which the order was created, along with a state proof and the associated state root, allowing the runtime to check whether the `OnDemandOrderPlaced` event was emitted.

However, the issue with this approach is that the runtime cannot verify whether this block was actually part of the relay chain. To address this, ancestry proof would be required. To validate the block provided by the collator, the runtime would need access to the hash of the current block. This would require a modification to the `RelaychainStateProvider` by introducing a new function that returns the current block hash.

Manual Reward Claiming

The order service tracks relay chain events, and when it detects an `OnDemandOrderPlaced` event, it stores the details in memory and provides inherent data to reward the order placer. However, the block author could theoretically omit this step and choose not to include the inherent data, preventing the order placer from receiving a reward. While there is no clear incentive to do this, it remains a possibility.

To prevent this, the order placer should have the ability to call an extrinsic and provide proof that they placed an order when expected but did not receive a reward. In such cases, the pallet should verify the proof and reward the order placer accordingly.

Worth noting that "manual" here does not mean that an actual person should claim the reward. Rather, it means that the collator, upon detecting that it did not receive a reward, will claim manually.

More Comprehensive Testing

The current e2e test is fairly basic; however, it covers the main functionality and ensures it works. For the modules to be production-ready, more comprehensive testing is needed to cover edge cases and ensure the modules function as expected.