Transaction Concept

Task/Group -> postgredb -> tasksqueue -> taken & scheduled by RTS -> wrote in Redis DB -> send to dispatcherqueue -> bearbeitet von dispatcher -> feedback in feedbackqueue

Task -> postgredb -> prioqueue -> bearbeitet von dispatcher -> feedback in feedbackqueue

Schritt 1: Senden von TaskApi in die TaskQueue:

Hier können keine RaceConditions auftreten, da die Queue die Anfragen „sequentalisiert“

Schritt 2: Entnehmen von Task durch Scheduler Instance(normal):

Transaction auf scheduleTask Ebene, da: Bei zwei Schedulern, die beide aufgrund eines neuen Tasks in der Taskqueue getriggert werden, werden beide alle Tasks die in der Redis db (plus den neue) enthalten sind schedulen, somit kommt es zu dopplungen in der dispatcherqueue -> muss als Transaction ablaufen

Eine Idee wäre auf Objekt ebene zu sperren. Ein Scheduler würde dann die objekte die er sich holt bis zum ende seiner Transaktion sperren und dann wieder releasen. Somit könnten die anderen Scheduler in dieser Zeit neu hinzugekommene Objekte weiterhin schedulen. Hätte aber den nachteil das die neu hinzugekommenen Objekte sofort weiter verabeitet werden

Schritt 2: Entnehmen von Task durch Scheduler Instance(prio):

Wird direkt von der Taskapi in in die prioqueue geschrieben -> keine Raceconditions, da Priority Queue die Anfragen „serialisiert“