Engineer

# File d’attente

travail mis en ordre pour être repris par un autre programme

## Ce que c’est

Une file d’attente est un espace de stockage intermédiaire entre deux programmes : l’un y dépose des messages ou des tâches, l’autre les retire lorsqu’il est prêt, généralement en commençant par les plus anciens. La plupart des systèmes utilisent plusieurs files, une par type de travail. Le programme qui ajoute du travail est le producteur, celui qui le prend est le consommateur, et ce qui conserve les messages entre les deux est le courtier de messages, qui peut être un serveur dédié ou une table dans une base que le système possède déjà.

Cette organisation évite aux deux côtés de s’attendre. Une requête web qui doit déclencher un courriel ajoute un message et répond aussitôt à l’utilisateur ; un processus de traitement récupère le message quelques instants plus tard et, si cent messages arrivent ensemble, les traite à son propre rythme. Une pointe de trafic devient du travail en attente, et une dépendance devient un délai.

Deux notions décrivent les garanties de livraison d’une file. La première est l’acquittement : un consommateur indique au courtier qu’il a fini de traiter un message, et le message n’est supprimé qu’à ce moment-là. Dans le modèle AMQP 0-9-1, que RabbitMQ implémente, un message dont le consommateur meurt avant de l’acquitter est transmis à un autre consommateur. La deuxième est la durabilité : une file durable contenant des messages persistants survit au redémarrage du courtier, contrairement à une file conservée uniquement en mémoire.

Trois notions voisines méritent d’être distinguées. Une file de messages transporte des données entre services. Une file de tâches transporte des unités de travail destinées à des processus d’arrière-plan, chacune avec ses arguments, un état et un nombre de tentatives. Un journal, le modèle d’Apache Kafka, ressemble de l’extérieur à une file, mais conserve les événements après leur

lecture pendant une durée de rétention choisie par sujet, ce qui permet à plusieurs lecteurs de rejouer le même historique. Lire dans une file supprime le message ; lire dans un journal ne fait qu'avancer la position du lecteur.

Une file ne nécessite pas forcément un produit distinct. PostgreSQL peut jouer ce rôle avec une table de tâches et une requête qui s'attribue la ligne suivante tout en sautant celles qu'un autre processus a déjà verrouillées. Le manuel cite les tables servant de files d'attente pour expliquer l'existence de la clause `SKIP LOCKED` :

```
WITH next AS (
  SELECT id
    FROM jobs
   WHERE state = 'available'
   ORDER BY id
   LIMIT 1
     FOR UPDATE SKIP LOCKED
)
UPDATE jobs
   SET state = 'running',
       attempt = attempt + 1
  FROM next
 WHERE jobs.id = next.id
RETURNING jobs.id, jobs.args;
```

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'une file d'attente ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des chiffres disponibles là où il faut, à la fréquence prévue, sans recopier un tableur à la fin du mois.

**Découvrir nos prestations** Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Un plan d'infrastructure pour 14 départements — 14 départements de recherche sur un même plan d'infrastructure

**Un plan d'infrastructure pour 14 départements** Contactez-nous →

## Pourquoi c'est important

Une file décide ce qui fait attendre l'utilisateur. Le travail qui y est déposé se déroule après l'envoi de la réponse : une page qui déclenche un courriel, un PDF, une réindexation de recherche et un recalcul peut donc répondre dans le temps nécessaire pour écrire quatre lignes, les quatre tâches lentes se terminant

ensuite. Pour un acheteur, c'est la différence entre une validation de commande qui répond en quelques centaines de millisecondes et une autre qui répond lorsque le serveur de messagerie est prêt.

Elle décide aussi de ce qui survit à une panne. Un travail lancé dans un thread d'arrière-plan vit dans la mémoire de ce processus, et un déploiement qui redémarre ce processus fait disparaître le travail, sans laisser de trace du fait qu'il devait être exécuté. Un travail dans une file durable constitue au contraire un enregistrement : il peut être réessayé, examiné et compté.

Elle sépare également deux rythmes. Les producteurs avancent au rythme du trafic entrant ; les consommateurs à celui du service le plus lent qu'ils appellent. Une file entre les deux absorbe l'écart, et sa profondeur devient l'un des indicateurs les plus fidèles d'un système. Une file qui grossit depuis une heure indique que les consommateurs prennent du retard, quoi que suggère le graphique du processeur.

Les avantages ont un coût. Le travail reste désormais invisible entre son envoi et son exécution. La livraison se fait presque toujours au moins une fois : un consommateur doit donc tolérer de voir deux fois le même message. L'ordre est moins garanti qu'il n'y paraît. Et un courtier dédié est un serveur supplémentaire à exploiter, sécuriser, surveiller et mettre à niveau.

## À quoi cela ressemble en pratique

Sur une plateforme, l'envoi de courriels, la reconstruction d'un index de recherche, la génération de documents et le recalcul des classements ont tous été retirés du chemin des requêtes. Le travail confié à une goroutine disparaît lorsque le processus redémarre au milieu d'un déploiement : il a donc été placé dans une file de tâches River conservée dans PostgreSQL. La base était déjà la source de référence du système, ce qui permet d'ajouter une tâche dans la même transaction que les données concernées. La plateforme y a gagné la possibilité de réessayer et un historique consultable ; la limite annoncée est qu'une file intégrée à la base ne pourra pas monter en charge indéfiniment.

Dans une application macOS construite sur une bibliothèque C, la file servait à préserver l'ordre. La boucle d'événements bloquante de la bibliothèque émet des événements dans un ordre important, car une modification doit arriver après le message qu'elle modifie. Lancer une tâche par événement laisse l'environnement d'exécution les ordonnancer librement : une modification peut

donc dépasser le message auquel elle se rapporte. La conception livrée comporte un thread producteur, un consommateur et une file entre les deux, puis un unique passage sur l'acteur principal. L'ordre tient parce qu'il n'existe qu'un seul chemin. La contrepartie concerne le parallélisme : chaque événement passe par cet unique consommateur.

Une file peut aussi nuire. Dans une application macOS, une rafale d'événements de journalisation du noyau applicatif était conservée dans un flux principal non borné et copiée vers des abonnés dont les consommateurs n'utilisaient pas ces événements. Les rapports de diagnostic ont relevé 111 Go de pages compressées sur une machine de 36 Go ; un échantillon distinct de l'empreinte mémoire a relevé une croissance d'environ 41 Mo par seconde pendant une boucle de tentatives de connexion. Le chemin des événements a reçu des limites explicites, avec des abonnements filtrés par type d'événement et une politique explicite de débordement. Les événements de diagnostic répétitifs peuvent être abandonnés et l'état concerné actualisé plutôt que de conserver un nombre illimité d'événements. Des contrôles d'architecture et des tests de respect des budgets de ressources de l'application défendent cette conception, mais le dossier ne fournit ni mesure de mémoire après correction ni benchmark de production.

## Où les problèmes surviennent

Les problèmes de file apparaissent tard et ailleurs, car le travail est invisible par conception.

**Le travail en attente augmente jusqu'à épuiser la mémoire ou le disque.** Une file sans limite accepte du travail plus vite qu'il n'est exécuté et conserve la différence. Donnez à chaque file une longueur maximale et décidez à l'avance de la conduite à tenir à cette limite : ralentir le producteur, refuser le nouveau travail ou abandonner le plus ancien. Surveillez ensuite deux nombres, la profondeur et l'âge du message le plus ancien, car une file courte de messages très anciens est aussi problématique qu'une longue file.

**Le même message est traité deux fois.** Un consommateur termine le travail et meurt avant de l'acquitter ; le courtier livre de nouveau le message, comme il le doit. La documentation de RabbitMQ demande aux consommateurs de prévoir ces nouvelles livraisons et d'être conçus pour l'idempotence. Donnez un identifiant à chaque message et enregistrez ceux déjà traités dans la même

transaction que le travail lui-même. Une note du site sur la vente de stocks limités sans survente construit son mécanisme d'écriture autour de cette même hypothèse.

**Les messages arrivent dans un ordre différent de leur envoi.** Les consommateurs parallèles, la prélecture et les nouvelles tentatives qui replacent un message en échec derrière les plus récents réordonnent tous le travail. Lorsque l'ordre compte, faites passer tout ce qui concerne un compte ou une commande par la même partition ou le même consommateur. Kafka le fait par clé, au prix d'une limite : une clé très sollicitée ne peut avancer plus vite qu'un seul consommateur.

**Un message défectueux bloque le traitement ou boucle indéfiniment.** Un message dont la charge utile fait planter le consommateur à chaque fois est remis en file, livré de nouveau et provoque un nouveau plantage. Limitez les tentatives et déplacez les messages qui les ont toutes épuisées dans un endroit distinct où une personne pourra les examiner. RabbitMQ appelle cela dead-lettering et l'applique aux messages rejetés, expirés ou qui dépasseraient la limite de longueur.

**Le travail disparaît au redémarrage.** Un canal, une liste en mémoire ou un thread d'arrière-plan est une file qui ne conserve rien après un redémarrage. En AMQP, une file durable ne suffit pas non plus : les messages doivent également être publiés comme persistants. Décidez pour chaque type de travail si sa perte est acceptable.

**La base et la file divergent.** Un service écrit une ligne, publie ensuite un message et plante entre les deux : l'un existe alors sans l'autre. Conservez la file dans la même base pour qu'une tâche ne devienne visible des processus de traitement qu'au commit de sa transaction, comme le fait River, ou écrivez le message dans une table d'envoi au cours de cette transaction et publiez-le à partir de là.

## Pour aller plus loin

- Le modèle AMQP 0-9-1 expliqué — échanges, files, liaisons et acquittements, présentés par RabbitMQ.
- Acquittements des consommateurs et confirmations des producteurs — quand un courtier considère un message comme livré et pourquoi il peut être livré de nouveau.

- Échanges de messages non livrables — les conditions qui écartent un message et sa destination.
- La clause de verrouillage de SELECT — `FOR UPDATE` et `SKIP LOCKED`, ainsi que les tables servant de files auxquelles ils conviennent.
- River : premiers pas — une file de tâches qui conserve ses tâches dans la base de l'application elle-même.
- Apache Kafka : introduction — sujets, partitions et rétention dans le modèle du journal.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

**Consulter les tarifs** Contactez-nous →

## Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'une file d'attente ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

**Découvrir nos prestations**

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/queue/