バッチジョブと一つずつ

今日、私たちのアパートで水道に問題が発生し、しばらく水が使えなくなりました。食事を終えて、家族と私が残した山積みの食器を見て、いくつかの疑問が浮かびました。

一つは、給水がなくても食洗機が動作し続ける方法です。バケツの水に接続できるように設計されているかもしれません。さらに、水道管の取り付けヘッドは柔軟性があり、公共の水道から自家製のバケツの水に簡単に切り替えられるようにする必要があります。

もう一つのポイントは、仕事をバッチで行うか、一つずつ行うかという問題です。私たちは毎食後に皿を洗うこともできますし、1 日後や数日後にまとめて洗うこともできます。これは、私たちが食事をし、皿を洗うという視点からのものです。また、食洗機が一度にどれだけの皿を収容できるかという視点からも、この問題にアプローチすることができます。

それはプログラミングを思い出させます。私たちはタスクを一括で実行することも、一つずつ実行することもできます。

バッチ処理を行うと、一つの明白な問題が生じます：それはより多くのリソースを必要とすることです。洗い物を遅らせることでより多くの食器が必要になり、データを蓄積して処理を遅らせることでより多くのメモリスペースが必要になります。

現実世界では、一度に扱えるスペースやアイテムの数には限界があります。例えば、食器洗い機はおそらく最大で 20 枚の食器しか洗えません。これは、プログラムがコンピュータ上で使用できるメモリに制限があるのと同様で、道路が通過できる車の数に制限があるのと同じです。

また、ジョブをどのように分割するかという問題もあります。1 アイテムずつ分割すべきか、それとも 3 アイテムずつ分割すべきか？

料理や車の場合、各アイテムを 1 つの単位として扱うのは簡単です。つまり、料理は料理であり、車は車です。通常、これらはそれ以上小さな部分に分解することはできません。ただし、例外もあります。例えば、多くの車を運ぶ大型トラックの場合、1 つの大きな単位と、道路を通過する多くの車に分解することができます。

プログラミングにおいては、はるかに柔軟性があります。挿入や更新の SQL でさえも、より小さな部分に分解することができます。ダウンロードジョブ、DFS 検索、クエリなどは言うまでもありません。

OK、これで処理単位について考えました。次に問題となるのは、1 バッチで何単位を処理すべきかです。1 単位から総単位数の間の任意の数が考えられます。

ここでの問題は、ジョブのバッチ数を固定するか、動的にするかということです。生成 AI の場合、入力テキストの総文字数は柔軟です。コンテキストの制限や入力の制限はありますが、その範囲内では柔軟に対応できます。

食洗機を使用する際、その内部空間には限界があります。その限界の中で、洗う皿の数は柔軟に調整できます。通常、洗う必要のある皿をできるだけ多く機械に入れます。

プログラムにとって、データベースが一度に処理する SQL のバッチには制限があります。その制限内では、処理できる SQL の数は柔軟です。しかし、クライアントからデータベースサーバーに SQL を渡すネットワークタスクや、ユーザーがどれだけ待てるか、そしてバッチ内の一つの単位タスクが失敗した場合に何が起こるかについても考慮する必要があります。

したがって、1 つのバッチでどれだけの単位タスクを行うべきかという問題については、ジョブの目的、下流の消費者やハンドラーの制限、そして失敗の確率を考慮する必要があります。

この考え方は多くのことに応用できます。基本的に考慮すべき問題は 2 つあります：ユニットタスクとは何か、そして 1 バッチでいくつのユニットを処理すべきかです。これらの問題を考えることで、最適な解決策にたどり着くことができるかもしれません。