OS の TCP パラメータをチューニングし、パフォーマンスを高める

ネットワーク　OS　性能　仕組み・原理

■　目的

TCP の実装においては、Window と Window Scale というパラメータがフロー制御に用いられ、ack 応答の効率化に寄与している。一方通信時のネットワーク容量超えを減らし、輻輳発生を避けるために、スロースタートメカニズムが用意されている。これらは、安定的な通信を保証する TCP においては、なくてはならないものである。しかしながら、通常の利用ではこの両者をほとんど意識することがない。したがって、この 2 つの機能が実際の通信においてどのように働いているのかが、正確に理解されることは少ない。

上記の機能を司るパラメータのいくつかは、カーネルパラメータとして用意されている。したがって、今あるネットワーク環境に適切な値を設定すれば、TCP 通信のパフォーマンスが高まることになる。その技術を習得すれば、むやみにスケールアップやスケールアウトをせずとも、パラメータ設定のみによって TCP 通信のパフォーマンスが向上し、コスト削減につなげることも可能になる。

本項目では、上記 TCP の 2 つの機構（フロー制御と輻輳制御）の振る舞いを、実際にカーネルパラメータの値を変化させながら実験を繰り返すことによって、理論と実践の両方から理解し、TCP 通信のパフォーマンス向上にどの様に寄与するか（しないか）を検証する。

■　前提

とくになし。

■　課題

1. **フロー制御について、以下の観点からレポートする。**

    - Window と Window Scale というパラメータの役割はなにか。
    - ack を返すタイミングはどの様に計算されるか。またどの様に効率化できるか。
    - ack 応答を効率化することのメリットは何か。

2. **輻輳制御について、以下の観点からレポートする。**

    - なぜスロースタートメカニズムは必要とされているのか。
    - スロースタートメカニズムはどのように実現されているか。
    - スロースタートメカニズムが輻輳制御に貢献するのはなぜか。

3. **上記の理解のもと、どのようなカーネルパラメータをどのように変更すればパフォーマンスが向上するか（または、しないか）をあきらかにする。**
4. **与えられたネットワーク環境で実際に通信をおこない、その効果を測定する。**

    - 検証ツールを用意して、実際に効果測定する。
    - この実験結果をリポートする。

■ 留意点

現在のように安定したネットワーク環境が用意されていても、この仕組みは必要なのかどうか、を考察できると良いと思う。結果、TCP 不要（UDP オンリー）、という見解も成り立つかもしれない。また、昨今広まってきている QUIC の理解につながるとよい。HTTP/3 も UDP+QUIC の上に HTTPS が実装されることになっているので。

IPv4 と IPv6 での実装の違い、カーネルパラメータの違いについてもレポートし、実験できることが望ましいが、それは余裕のある受講者向けの課題としておく。

超アドバンストコースとして、実際にカーネルのネットワークスタックを改造しながら実験を繰り返すということも考えられる。

また、BBR や CUBIC などの輻輳制御アルゴリズムの適切な選択、というものも課題として存在しうる。それは本項目の課題とするのではなく、別の課題項目として考えることにする。

■ 参考

➢ TCP 帯域幅（bandwidth）関連パラメータのキーワード

- TCP window scaling
- TCP socket buffer size
- congestion window size （congestion window サイズは、アプリケーションやカーネルパラメータで設定はできないはずである）

➢ 具体的なカーネルパラメータ

- net.ipv4.tcp_window_scaling
- net.core.rmem_default
- net.core.wmem_default
- net.core.rmem_max
- net.core.wmem_max
- net.ipv4.tcp_rmem
- net.ipv4.tcp_wmem
- net.ipv4.tcp_mem