エッジコンピューティングにおけるマイクロサービスアーキテクチャに基づくオンラインゲームアーキテクチャの提案と評価

2018SE048 村木俊斗　2018SE057 成田裕太
2018SE077 佐々木伸一
指導教員：宮澤元

1　はじめに

近年ネットワークに接続してゲームを行うオンラインゲームが普及してきている．ゲームをプレイするユーザはクラウドに存在するゲームサーバプログラムに接続することで様々なゲームを楽しむことができる．オンラインゲームでは，ユーザ同士で対戦を行ったり，協力を行うことができる．

オンラインゲームのゲームサーバは，サービス全体を一つのプログラムとして作り上げるモノリシックなアーキテクチャに基づき作成されることが多い．モノリシックなアーキテクチャに基づき作成されたゲームサーバでは，ゲームの処理が単一のサーバプログラムに集中することによる処理の遅延が問題となることがある．オンラインゲームでは多くの処理を接続した一つのゲームサーバで行っている．一つのゲームサーバは複数のクライアントから接続を受けつけていて，そこで発生するすべての処理を行っている．その結果，一つのゲームサーバに多くの処理が集中してしまい，処理による遅延が発生してしまう恐れがある．

モノリシックなアーキテクチャに基づくサービスの問題を解決するために，マイクロサービスアーキテクチャを用いた開発方法が提案されている [1]．実際にマイクロサービスアーキテクチャを用いることで処理による遅延を解決した事例も報告されている [2]．この方法をオンラインゲームのゲームサーバの開発に用いると，一つのゲームサーバに集中するゲームの処理を複数のサーバに分散することができる．これにより一つのゲームサーバに集中していた処理を分散させることで処理による遅延を抑えることができる．

マイクロサービスアーキテクチャに基づき作成されたゲームサーバでは，通信による遅延が増えてしまう問題がある．マイクロサービスアーキテクチャに基づくゲームサーバは互いに通信しあう複数のサーバプログラムから構成される．その結果，モノリシックなアーキテクチャに基づくゲームサーバでは発生しなかったサーバプログラム間の通信による遅延が発生してしまう．

本研究の目的は，遅延の問題を解決できるオンラインゲームアーキテクチャを開発することである．ゲームサーバへの処理を分散させ処理による遅延を解消するとともに，エッジコンピューティングを活用して通信遅延を軽減する．

本研究では，オンラインゲームサーバにおける処理による遅延を解消するために，ゲームサーバをマイクロサービスアーキテクチャに基づきマイクロサービスに分割する．具体的なサービス分割とエッジサーバへの配置を決定するにあたって，既存のオンラインゲームの分析に基づき，サービス間およびサービスとユーザ間の通信回数や通信遅延を考慮して，以下の 3 つのアーキテクチャを提案する．

- 低処理量向けアーキテクチャ
- 多接続向けアーキテクチャ
- 高処理量向けアーキテクチャ

提案アーキテクチャに基づき実装されたオンラインゲームを用いて，提案アーキテクチャの有効性を評価する．評価項目は各サーバの CPU 使用率，CPU 時間と通信時間である．CPU 使用率，CPU 時間を調べることで各サーバの負荷を計測することができ，通信時間を調べることでどの程度通信遅延が発生するか調べることができる．これらの項目について提案するアーキテクチャとモノリシックなアーキテクチャで作成したオンラインゲームを比較して，提案する新たなオンラインゲームアーキテクチャが有効であるのか評価を行う．

本研究の研究課題は新しいオンラインゲームアーキテクチャの提案と，提案したアーキテクチャの評価の 2 点である．

2　研究の背景

本節では，本研究の背景としてオンラインゲームの現状についてと，マイクロサービスアーキテクチャ，エッジコンピューティングのそれぞれの内容について述べる．

2.1　オンラインゲームの現状

オンラインゲームとはユーザがインターネットに接続された機器を用いて複数のユーザとともにゲームをプレイできるものである．ゲームの種類は様々であり，1 対 1 の格闘ゲームから多くのユーザが接続して探索などを行える MMORPG まで多岐にわたる．リアルタイムで複数のユーザと共同作業を行ったり，情報共有をすることができることが特徴である．

モノリシックなアーキテクチャを用いて作成されたゲームサーバの利用例を以下に示す．ユーザ側であるクライアントからゲームサーバに接続要求を行い，ユーザは様々なゲームをプレイすることが可能となる．接続されたゲームサーバはユーザからの入力やプレイしているゲームの状況変化に応じて適当な処理を行い，その結果をユーザ側であるクライアントに送信する．

モノリシックなアーキテクチャで作成されたゲームサー

バは単一のプログラムとして実現されているので，プログラムの処理を高速に行うことができるという利点を持つ．一方で，一つのゲームサーバに多くのクライアントが接続されてしまうと処理の集中による遅延が発生してしまう．

今回分析したオンラインゲームについてモノリシックなアーキテクチャで作成すると以下のような問題点が生じる．

- 格闘ゲーム
  頻繁に処理が行われサーバプログラムで行う処理が集中して遅延が発生しまう．
- MMORPG
  大人数が同時にサーバに接続しておりそのせいでサーバに処理が集中してしまう．それによりサーバで行われる処理が重くなり遅延が発生してしまう．
- シューティングゲーム
  サーバで多数の情報の処理を行わなければならず，サーバに大きな負荷がかかり遅延が発生してしまう．

2.2 マイクロサービスアーキテクチャ

マイクロサービスアーキテクチャとは，複数の規模の小さいサービスを組み合わせて単一の大きなアプリケーションを構成するというソフトウェア開発の手法の一つである．分割したサービス一つ一つが独立したサービスとして起動し，相互に通信を行うことでユーザにシステムを提供している [1]．

マイクロサービスアーキテクチャを用いて作成されたゲームサーバの利用例を以下に示す．ユーザ側であるクライアントからゲームサーバに接続要求を行い，ゲームをプレイすることが可能となる．接続されたゲームサーバはユーザからの入力やプレイしているゲームの状況変化に応じて適当な処理を行い，その結果をユーザ側であるクライアントに送信する．この際，ゲームサーバは各々独立したサーバの集合として構成されていて，各サーバはそれぞれ与えられた処理を行っている．各サーバは他のサーバと互いに通信をして情報の共有を行っている．ゲームサーバ内で処理を分散させることができるので処理による遅延を低減することができる．

マイクロサービスアーキテクチャで作成されたシステムの実際の事例として Gunosy というアプリが存在する．Gunosy ではニュース配信時にかかる急激な負荷によって処理による遅延が発生していた．そこでマイクロサービスアーキテクチャを用いて問題を解決したという事例がある [2]．

2.3 エッジコンピューティング

エッジコンピューティングとは，ユーザに近いエリアのネットワークにサーバを分散配置して処理を行うコンピューティングモデルの一つである．エッジコンピューティングを用いることでデータを収集した機器や，その機器に近いエリアでデータを処理することで，処理の負荷を分散することができる．またユーザに近い位置で通信を行うので，通信による遅延を減らすことができる．

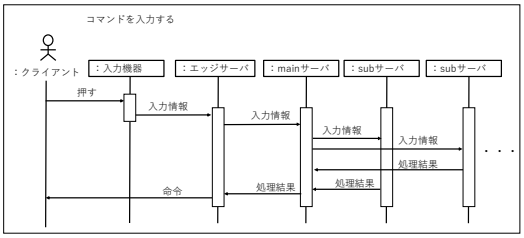

図 1 低処理量向けアーキテクチャの配置例

3 提案するオンラインゲームアーキテクチャ

1 節で述べた問題を解決するためにマイクロサービスとエッジコンピューティングを用いたオンラインゲームアーキテクチャを提案する．これから提案する 3 つのアーキテクチャはそれぞれ既存のゲームから着想を得ている．

3.1 低処理量向けアーキテクチャ

このアーキテクチャはサーバプログラムで行う処理を**図 1** のように main サーバと sub サーバで分けている．main サーバではゲームを進行する上で必要な処理を行う．sub サーバではデータの更新頻度が少ない処理やクライアントからの入力が関係ない処理を行う．またこのアーキテクチャでは 1 つの main サーバに複数の sub サーバが接続している．これによりクライアントの入力で発生する処理によるサーバ間の通信回数を減らすことができる．これにより頻繁に処理が行われるゲームでサーバが行う処理が集中して遅延が発生しまうのを防ぐことができる．

3.2 多接続向けアーキテクチャ

このアーキテクチャでは行う処理に必要な情報の範囲と種類をもとに，エッジサーバを含む複数のレベルの部分管理サーバと全体管理サーバとでサービス分割を行う．この分割方法では**図 2** のようにクライアントに近い部分管理サーバほど細かい情報の処理を行う．逆に全体管理サーバに近づけば近づくほど大まかな情報の処理を行う．部分管理サーバは全体管理サーバから広がり，探索木のように接続している．これにより大人数が接続してもプレイヤーなどの情報の処理を特定の範囲ごとに効率的に行うことができる．これにより多接続によって発生した処理が集中してしまい，サーバで行われる処理が重くなり遅延が発生してしまうのを防ぐことができる．

3.3 高処理量向けアーキテクチャ

このアーキテクチャでは**図 3** のようにゲームサーバで行う処理のそれぞれを担当する divided サーバにサービスを

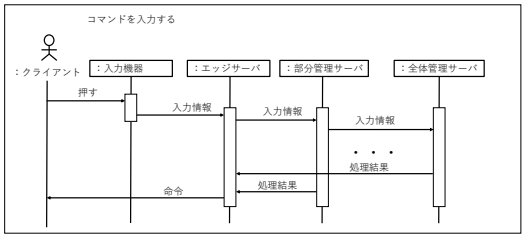

図 2 多接続向けアーキテクチャの配置例

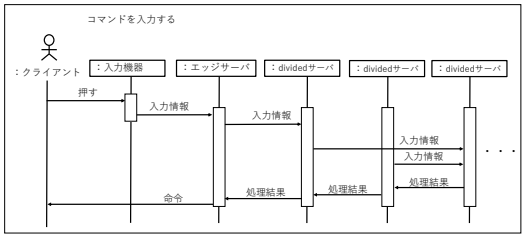

図 3 高処理量向けアーキテクチャの配置例

分割する．多数の情報の処理を分割した divided サーバに割り当てることでサーバの負荷が分散される．これによりサーバで多数の情報の処理を行うゲームでサーバに大きな負荷がかかり遅延が発生するのを防ぐことができる．

4 提案アーキテクチャに基づくオンラインゲームの実装

提案アーキテクチャの有効性を評価するために，提案アーキテクチャに基づくオンラインゲームとして，トランプのスピードゲームを実装した．スピードはやり取りするデータ量が少なく，通信頻度が高いゲームである．3 節で述べた提案アーキテクチャに加え，比較対象としてモノリシックアーキテクチャに基づくプログラムも実装した．本来，プレイヤー側が操作しているマシン上ではクライアントプログラムの処理のみ行われるのが普通だが，今回エッジサーバで動作させるプログラムのデータ量が少ないのでエッジサーバに行わせる処理をクライアントプログラムに行わせている．実装は全て C 言語を用いて行った．

5 実験

提案アーキテクチャの有効性を確認するために実験を行う．

5.1 実験環境

AWS(Amazon Web Services) の Amazon EC2(Amazon Elastic Compute Cloud) を利用し，インスタンス (仮想サーバ) 内でサーバプログラムを動かし実験を行った．クライアントプログラムはノート PC 上で動かした．インスタンスは最大 4 つとノート PC2 台を利用する．利用するインスタンスのタイプは t3.small である．実験環境の詳細を**表 1** に示す．

表 1 実験環境の詳細

	インスタンス	ノート PC
CPU	Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
OS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS (WSL)
メモリ	2GiB	8GiB

5.2 実験内容

4 種類のオンラインゲームアーキテクチャに基づき試作したゲームを用いて計測実験を行う．各アーキテクチャのサーバの経過時間・CPU 時間とインスタンスの CPU 使用率，クライアントの経過時間について計測した．サーバの経過時間と CPU 時間は他サーバ（クライアント）から情報を受信してから送受信などの処理を行うまでの時間を計測している．CPU 使用率はゲームを開始してから終了までを計測している．クライアントの経過時間はサーバに情報を送信してから受信するまでの時間を計測している．経過時間と CPU 時間の計測では clock_gettime 関数を用い，CPU 使用率の計測では Prometheus[3]，Node Exporter，Grafana[4] を用いた．また，経過時間と CPU 時間はそれぞれ 50 回，CPU 使用率は 1 秒間隔で計測し，平均値を求めた．

5.3 実験結果

サーバの経過時間と CPU 時間とインスタンスの CPU 使用率の実験結果をアーキテクチャごとにまとめた．また，クライアントの経過時間もまとめた．経過時間と CPU 時間は有効数字 2 桁，CPU 使用率は有効数字 3 桁でまとめた．

5.3.1 モノリシックなアーキテクチャ

サーバの経過時間・CPU 時間とインスタンスの CPU 使用率の平均を**表 2** に示す．経過時間が 18ms，CPU 時間は 8.3μs，CPU 使用率は 2.68% という結果になった．

表 2 計測結果 (モノリシック)

	サーバ
経過時間 [ms]	18
CPU 時間 [μs]	8.3
CPU 使用率 [%]	2.68

5.3.2 低処理量向けアーキテクチャ

各サーバの経過時間・CPU 時間とインスタンスの CPU 使用率の平均を**表 3** に示す．全体管理サーバの方がカード変更確認サーバよりも経過時間と CPU 時間と CPU 使用率が大きいという結果になった．すべてのサーバはモノリシックなアーキテクチャより経過時間と CPU 時間が短い．CPU 使用率は全体管理サーバの方が高い．

表 3 計測結果 (低処理量)

	カード変更確認サーバ	全体管理サーバ
経過時間 [μs]	2.8	38
CPU 時間 [μs]	2.5	6.1
CPU 使用率 [%]	2.60	2.74

5.3.3 多接続向けアーキテクチャ

各サーバの経過時間・CPU 時間とインスタンスの CPU 使用率の平均を**表 4** に示す．場の管理サーバの方が経過時間と CPU 時間の平均値が大きいという結果になった．CPU 使用率はクライアント管理サーバ 1 が最も高いが，大きな差はない．経過時間と CPU 時間はモノリシックなアーキテクチャより短い．しかし，CPU 使用率は多接続向けアーキテクチャの方が高い．

表 4 計測結果 (多接続)

	場の管理サーバ	クライアント管理サーバ 1	クライアント管理サーバ 2
経過時間 [μs]	38	2.8	2.8
CPU 時間 [μs]	4.7	1.8	1.8
CPU 使用率 [%]	2.72	2.76	2.73

5.3.4 高処理量向けアーキテクチャ

各サーバの経過時間・CPU 時間とインスタンスの CPU 使用率の平均を**表 5** に示す．経過時間と CPU 時間は中継サーバが他のサーバより大きい．しかし，CPU 使用率は膠着確認サーバが 1 番高い．すべてのサーバはモノリシックなアーキテクチャより経過時間と CPU 時間が短い．また，CPU 使用率は膠着確認サーバだけモノリシックなアーキテクチャより高い．

表 5 計測結果 (高処理量)

	中継サーバ	膠着確認サーバ	場の更新サーバ	カード変更確認サーバ
経過時間 [μs]	56	4.4	4.3	31
CPU 時間 [μs]	6.2	0.85	4.1	4.1
CPU 使用率 [%]	2.65	2.81	2.51	2.67

5.3.5 クライアント

アーキテクチャごとの経過時間の平均を**表 6** に示す．モノリシックなアーキテクチャが最も時間がかかっている．他のアーキテクチャより約 50ms 経過時間が長い．

表 6 計測結果 (クライアント)

	モノリシック	低処理量	多接続	高処理量
経過時間 [ms]	73	26	23	24

5.4 考察

今回の実験結果から，提案アーキテクチャを用いるとモノリシックアーキテクチャを用いて実装したゲームと比べて，処理による遅延を抑えることができ全体的な通信遅延時間を抑えることができることがわかった．CPU 使用率に関してはアーキテクチャが異なっていてもあまり差が見られなかった．CPU 時間を見ると，どのアーキテクチャも CPU をあまり用いてないことがわかる．したがって今回の実験では CPU に対する負荷が足りず，提案アーキテクチャとモノリシックなアーキテクチャで CPU 使用率に関して優劣はつけられない．

6 おわりに

本研究ではマイクロサービスアーキテクチャに基づく新たなオンラインゲームアーキテクチャを提案した．提案アーキテクチャの有効性を確認するために，提案アーキテクチャに基づきオンラインゲームを試作し，実験を行った．実験の結果，サーバの経過時間とクライアントの経過時間はモノリシックなアーキテクチャが最も時間がかかっている．しかし，サーバの CPU 時間と CPU 使用率はどのアーキテクチャも明確な差はなく，実験に用いた試作ゲームの CPU 負荷が低かったことが原因であると考えられる．

今後の課題は，提案アーキテクチャの有効性の検討をさらに進めることである．特に，提案アーキテクチャそれぞれの有効性を検証するために，MMORPG や FPS などの CPU 負荷が高いゲームを含む様々なゲームを用いて評価実験を行う必要がある．また，アーキテクチャを提案するうえで各サービスの独立性や同期処理についても考慮する必要がある．

参考文献

[1] James Lewis, Martin Fowler, “Microservices,” https://martinfowler.com/articles/microservices.html (2021/12/28 access).

[2] 株式会社エスキュービズム，“事例つき！マイクロサービスとは？基礎知識からメリットとデメリットまでを解説,” https://ec-orange.jp/ec-media/?p=23458#no6 (2022/1/27 access).

[3] “Prometheus - Monitoring system & time series database,” `https://prometheus.io/` (2021/12/30 access).

[4] “Grafana: The open observability platform — Grafana Labs,” `https://grafana.com/` (2021/12/30 access).