修 士 論 文

分散アルゴリズムフレームワークのための semi-passive replicationのコンポーネント実装

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

鈴木 貴之

2005年3月

修 士 論 文

分散アルゴリズムフレームワークのための semi-passive replicationのコンポーネント実装

指導教官　片山 卓也教授

審査委員主査　片山卓也 教授
審査委員　鈴木正人 助教授
審査委員　DEFAGO Xavier 特任助助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

210050 鈴木 貴之

提出年月: 2005 年 2 月

目 次

図 目 次

第1章　はじめに

分散システムは、プロセスの故障に弱いという特有の問題を持っている。この問題を解決するために分散システムの fault tolerance（耐故障性、故障許容範囲）が大切になる。この耐故障性を支えるものの１つとして冗長性がある。冗長性は、複製技術を実装した複製サービスや複製コンポーネントのよって実現される。一般的に冗長性を実現する複製技術は、大きく２種類に分けられる。Active replication と Passive replication である。この２つの複製技術の利点は、互いに補完し合っている [1]。

本研究では従来の複製技術である Active replication と Passive replication を組み合わせた複製技術 Semi-passive repliation を分散アルゴリズムフレームワーク（以下 Neko と呼ぶ）[2] に実装し、他の複製技術と性能比較することである。Semi-passive replication は X.défago らにより考案された複製技術である [3]。この複製技術の正当性は証明されているが、実装および性能評価はまだ、行われていない。

次に本研究の特色は、Neko を用いることである。複製という概念は直感的には理解しやすい。しかし、複製技術の実装となると困難である。それは分散システム上で複製サービスを行うことが、複製サーバーの中で動く複製プロセス集合に対し、保障された矛盾のない状態を維持することを要求するからである。この複製プロセス集合間での矛盾のない状態を共有維持できるものの１つとして Consensus problem（合意問題）がある。この合意問題が Failure detector（故障した可能性のあるプロセスの探知）や Reliable broadcast（メッセージの送受信方法）を要求している。つまり、複製という概念を実装することは複製技術だけを理解すれば良いということではない。複製技術を支える複数の概念が存在しており、これらの概念の理解とこれらの概念を実現しているコンポーネントが必要なのである。Neko の中には複製技術を実装するのに必要なコンポーネントが豊富に揃っている。これらのコンポーネントを利用することで複製技術の実装がスムーズにできると期待できる。また、Neko は分散アルゴリズムを実装したプログラムでシミュレーション環境と実ネットワーク環境の両方で性能評価ができるので、Semi-passive replication の性能評価の制度および、信頼性を向上させることが期待できる。

本稿では Semipassive replication の構造をどのようにして構築するか、その際に大切なことは何かということを話の中心においている。しかし、その前に複製技術の実装に不可欠な要素である Failure detector、Reliable broadcast、そして合意問題について触れなければならない。これらのことは、この研究で自分がただ単に Semi-passive replication の実装というプログラミング的な作業だけでなく、複製技術を実装するために多くの概念の理解習得の結果を示すためでもある。また、分散システム上で複製技術を実装することの難し

さを強調するためでもある。

まず、２章では本研究の背景、目的、動機について述べる。この章では特に複製技術の大切さについて述べる。

次に３章では、複製技術について述べる。この章では従来の複製技術である Active replication と Passivereplication の説明をはじめ、複製技術に欠かせない要素である合意問題、Failure detector、Reliable broadcast についても説明する。

そして、４章では Semi-passivereplication の概要について述べる。この章では、Semi-passive replication の複製サービス、Semi-passive replication を支えている要素はなにか？従来の複製技術との違いなど Semi-passivereplication の全体像について説明する。そして、本研究の特色である Neko についても説明する。

５章では、Semi-passive replication の実装について述べる。Neko の中で、どのようにして Semi-passive replication を構築するか、Semi-passive replication の実装をする段階でつまずいた点、アルゴリズムとプログラミングのギャップをどのようにして埋めていったかを述べる。

６章では、Semi-passsive replication の評価について述べて、最後に本研究にまとめる。

第2章　背景

背景　分散システムはプロセスの集合体であり、それぞれのプロセが通信リンクを通してメッセージ交換を行うことでシステム全体の信頼性の向上を提供している [4]。そして、分散システムは、集中型システムに比べて安価に全体の処理能力の高いシステムが構築できる。しかし、分散システムの大きな問題の１つとして、取るに足らない小さな故障によってシステムの能力が傷付くことである。例えば、１つのプロセスの故障がシステム全体の危機にさらす可能性もある。したがって分散システムの耐故障性が大切になる。この耐故障性を支えるものの１つとして冗長性がある。冗長性は、複製技術を実装した複製サービスや複製コンポーネントによって実現される。

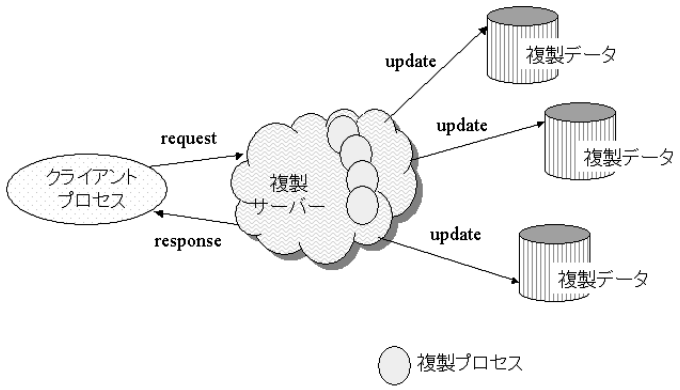

図 2.1: 一般的な複製サービスについて

動機　図 2.1 は、一般的な複製サービスの流れを表したものである。一般的な複製サービスでは、複製サーバーの中で動く複製プロセスが、複数のデータベースを管理、調整することでデータの冗長性を実現している。しかし、複数のプロセスが、クライアントからの１つ複製サービス要求に対して、同一の要求処理をするのでは、多くの計算処理コストを費やしてしまう。計算処理コストを多く費やす複製技術が分散システムの耐故障性を支え

る冗長性を実現できたとしても、それでは分散システムの利点を損なうことになる。したがって、計算処理コストが少なく、複製プロセスの故障が、クライアントに影響しない複製技術を実現することが大切である。

目的　従来の複製技術は大きく２種類に分けることができる。Active replication と Passive replication である。この２つの複製技術の利点は、互いに補完し合っている。Active replication と Passive replication を組み合わせることによって計算処理コストが低く、故障に強い複製技術が実現すると期待できる。X.Défago らが考案した Semi-passive replication は、従来の複製技術の利点をを組み合わせた複製技術である。Semi-passive replication では、従来の複製技術の利点を組み合わせるために特別な合意アルゴリズムを使っている。

本研究の目的は、Semi-passive replication のコンポーネントである特別な合意アルゴリズムと従来の複製技術のコンポーネントである合意アルゴリズムと比較して、Semi-passive replication の性能評価をすること。本研究で、Semi-passive replication のための特別な合意アルゴリズムと比較する合意アルゴリズムは、Chandra&Toueg の合意アルゴリズムである [5]。そして、Semi-passive replication のコンポーネントを Neko に実装することである。Semi-passive replication のコンポーネントを Neko に実装することで複製技術だけでなく、他の分散アルゴリズムを実装するときにそれらのコンポーネントが役に立つと考えられる。また、Neko の利便性も大きくなると期待できる。

第3章　複製技術について

ほとんどの分散システムが複製サービスを使用している。それは、サービスが必要とされる場所に複製サーバーを設置することによってシステム全体の処理能力を向上させることと同じくらいの耐故障性をサポートできるからである。しかし、分散システム上で複製サービスを実現することは困難である。その理由の１つとして、複製技術を実装する際に合意問題という大切な概念が関連しているからである。そして、もう１つの理由は、分散システムは故障に弱いという問題が複製技術を実装することを難しくしている。

複製技術を実現する複製プロトコルは一般的に２種類に分けることができる。集中管理プロセスを持たない Active replication と primary プロセスと呼ばれる集中管理プロセスを持つ Passive replication である。これら２種類の複製プロトコルは、互いの利点が補完しあっている。簡単に述べると、次のとおりである。

Active replication の利点

複製サーバーの中の複製プロセスの故障が複製サービスを必要とするクライアントプロセスに影響しないこと

Passive replication の利点

Active replication に比べてプロセスの計算処理資源が少なくて良いこと

この章では、最初に分散システム上に複製技術を実装する際に必要となる合意問題、Failure detector、Reliable broadcast について説明する。次に一般的な複製サービスが発生する時のイベント、イベントが使用するメッセージなど複製モデルについて説明する。ここで説明するイベント、メッセージは Active replication、Passive replication、そして Semi-passive replication での共通のイベント、メッセージである。この章以降の図には、ここで説明するイベント、メッセージを使う。次に複製技術の定義について述べる。この部分は Active replication、Passive replication ともに共通な定義である。この複製技術の定義に新たな定義を付け加えて様々な複製アルゴリズムができるのである。そして、Passive replication と Active replication の説明に移る。

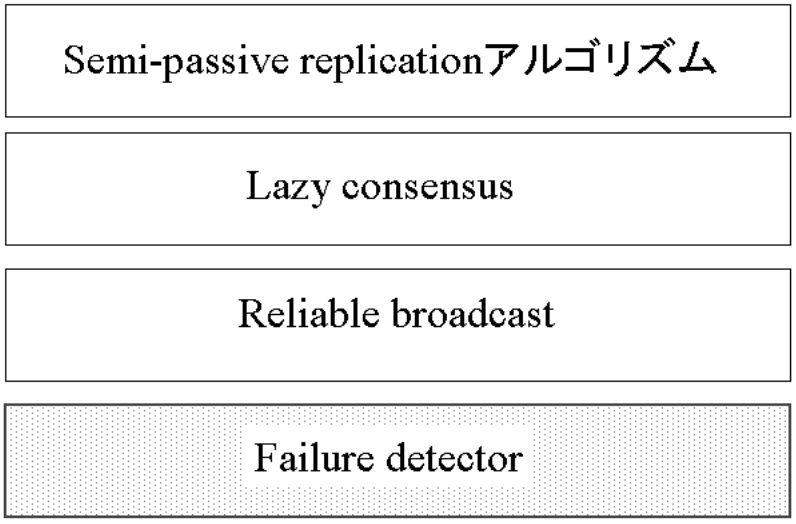

図 3.1: Semi-passive replication の枠組 1

3.1 Failure detector

プロセスの故障、ネットワーク分割、ネットワーク消失、２重のメッセージ送信などの故障が発生するかもしれない分散システム上で、プロセスが意思決定的に合意に達することは不可能である。それは、故障したプロセスが本当に故障したプロセスなのか、それとも単に応答スピードが遅いだけなのか区別することができないことが原因の１つである [6]。もし、分散システムの中で正確に動作しているプロセスが故障したプロセス情報を共有できたなら、合意問題は解決できるだろう。

ここでは、故障した可能性のあるプロセスを発見する Failure detector について説明する。Failure detector は、分散システム構築や分散アプリケーション作成の時には、必ず必要となる部分である。最初に基本的な２種類の Failure detector について説明する。図 3.1 は、Semi-passive replication の実装の枠組みを表している。図 3.1 で示すように Semi-passive replication の実装では Failure detector 層が最下層に位置する。最後に今回、Semi-passive replication の実装で用いた Heartbeat failuredetector について触れる。

3.1.1 Unreliable Failure detector

Unreliable Failure detector とは、合意問題を解決するのに十分な能力がある故障探知器のことである。(以下、◇S Failure detector と呼ぶ) この故障探知器は間違ってプロセスを故障したと疑う可能性がある。今回の Semi-passive replication の基本となっている Lazy consensus は、この ◇S Failure detector を使った Chandra&Toueg の合意アルゴリズムを変更したものである。◇S Failure detector は、次の要求によって定義される。

strong completeness

分散システムの中で故障する全てのプロセスは、正確に動作しているプロセスによって永久的に故障したものと疑われる

eventual weak accuracy

分散システムの中で正確に動作しているプロセスは、正確に動作しているプロセスを故障していると疑わない

3.1.2 Perfect Failure detector

多くの複製アルゴリズムが正確にプロセスの故障を探知できる Failure detector に依存している。(以下、*P* Failure detector と呼ぶ) つまり、多くの複製アルゴリズムは、*P* Failure detector に依存しているのである。同じ章で説明する Passive replication も *P* Failure detector を必要としている。それは、Passive replication が非同期システムでは、保障されていないからである。[1]

P Failure detector は、故障していないプロセスを故障したと疑わない故障探知機である。*P* Failure detector では、◇S Failure detector で述べた *completeness* と *strong accuracy* に次の *strong accuracy* の要求が必要となる。

strong accuracy

故障するまえに、故障したと疑われるプロセスはない

[1] この場合の非同期システムとは、プロセスの計算処理速度やメッセージの送信遅延の上限がないシステムのことである。

3.1.3 Heartbeat Failure detector

Heartbeat Failure detector とは、プロセスがハートビートメッセージを他のプロセスへ一定の時間幅でメッセージ送信することで failure dettector を実現する方法である。このハートビートメッセージが、制限時間の間に届かない場合は、Failure detector がプロセスが故障した可能性を探知するのである。Heartbeat Failure detector で大切になる時間とそれぞれの時間の関係は、次のようになる。

T_{send} ：ハートビートメッセージ送信時間、メッセージの送信時間幅

T_{out} ：制限時間、メッセージを受け取る制限時間

T_{tr} ：メッセージ伝送時間、メッセージがネットワークを通して伝わる時間

$alpha$:ネットワーク遅延時間

$T_{out} = T_{tr} + alpha$

実ネットワーク環境では、上記の式を基に Heartbeat Failure detector に関する時間の設定をしなければならない。しかし、Neko の中のシミュレーション実行では、メッセージ伝送時間はないので T_{send} と T_{out} の関係が大切になると考える。

以下には、制限時間の設定に Heartbeat Failure detector の性能が左右されることを示した。

制限時間が短い場合

プロセスの故障を素早く探知できる。しかし、間違った故障探知の可能性が高くなる。

制限時間が長い場合

間違った故障探知の可能性は低くなる。しかし、探知時間のコストが高くなる。

上記のように Heartbeat Failure detector の実装は二者択一となり制限時間は、ハートビートメッセージ送信時間を基に設定することになる。

3.2 Reliable broadcast

ここでは、Reliable broadcast と合意問題の関係に着目している。まず、メッセージをブロードキャストすることで引き起こす矛盾について説明する。次に Reliable broadcast のメリットは何かを考える。最後に Reliable broadcast と合意問題の関係について説明する。

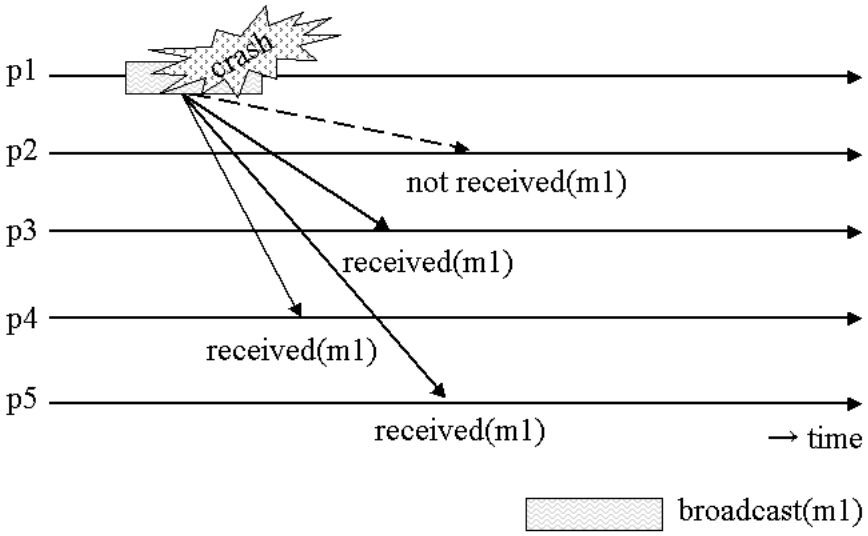

図 3.2: ブロードキャストが引き起こす矛盾の例

3.2.1 ブロードキャストが引き起こす矛盾

図 3.2 は、プロセスがメッセージをブロードキャストしているときに故障が発生したケースを示したものである。分散システム上で、通信リンクを通してメッセージ交換をしているプロセス集合がメッセージをブロードキャストしている間にプロセスの故障が発生したら、図 3.2 のように 1 部分のプロセス集合だけがメッセージを受け取る可能性が考えられる。

p3,p4,p5 ： メッセージを受け取るプロセス

p2 ： メッセージを受け取らないプロセス

このような矛盾は、分散システムの整合性の信頼を失うことになる。特に状態の共有を維持しなければならない複製サーバーにとっては、致命的である。

3.2.2 Reliable broadcast について

Reloable broadcast は、通信リンクの故障や悪意のないプロセスが存在する非同期システム上に簡単に実装することができる。また、Reliable broadcast では次の 3 つのことを保障している。

1. 正確に動作している全てのプロセスが持つメッセージ集合（ここでのメッセージ集合とは、正確に動作しているプロセスが送ったメッセージ集合を指す）は、同じである
2. 正確に動作しているプロセスがブロードキャストしたメッセージは届けられる
3. あやしいメッセージ（正確に動作していないプロセスがブロードキャストしたメッセージを指す）は、永久に届けられない。

特に複製プロセスが送るメッセージ集合の一致が大切となる合意問題では、上記の 3 つの特性が必要である。したがって、合意問題では Reliable broadcast が必要となる。ここで述べている 3 つの特性については、Reliable broadcast の特性で詳しく説明する。図 3.3 が表すように Semi-passive replivcation 実装の際には、Failure detector 層の上に Reliable broadcast 層が位置する。

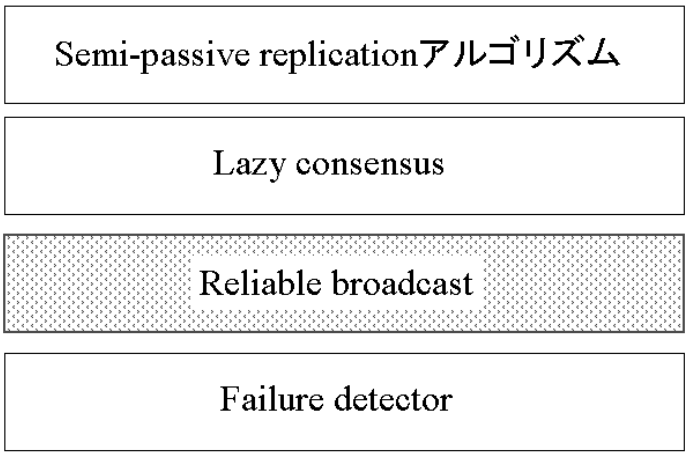

図 3.3: Semi-passive replication の枠組 2

3.2.3 Reliable broadcast によるプロセスのイベント

Reliable broadcast では、次の２つのイベントが発生する。

broadcast(m) ：プロセスが broadcst(m) を呼び出したら、メッセージ m をブロードキャストすることを表す

deliver(m) ：プロセスが deliver(m) を呼び出したら、メッセージ m を受け取ることを表す

ここでメッセージ m とは、プロセス集合がブロードキャストしたメッセージ集合の中のメッセージである。また、Reliable broadcast では受け取ったメッセージの順序、同一のメッセージ受け取り防止のためにブロードキャストされるメッセージには情報タグがつく。

ブロードキャストされたメッセージに付く情報タグ

sender(m) ：メッセージ送信者の識別情報

seq#(m) ：受け取ったメッセージ順序情報

情報タグの例　例えば、プロセス１ (p1) がメッセージ b と c を、この順番でブロードキャストしたことと、プロセス４ (p4) がメッセージ e をブロードキャストしたことを示す情報タグは次のとおりである。

メッセージ b ：sender(b)=p1, seq#(b)=1

メッセージ c ：sender(c)=p1, seq#(c)=2

メッセージ e ：sender(e)=p4, seq#(e)=1

3.2.4 Reliable broadcast の特性

Reliable broadcast は、次の３つの特性を満たしているブロードキャストである。

Validity（有効性）
正確に動作しているプロセスが broadcast(m) を呼び出すなら正確に動作している全てのプロセスは、結果的に deliver(m) を実行する

Agreement（合意性）
正確に動作しているプロセスが、deliver(m) を呼び出したなら正確に動作している全てのプロセスは deliver(m) を呼び出す

Integrity（整合性）

正確に動作していて、送信者の情報がはっきりしているメッセージがブロードキャストされた場合に限り、正確に動作している全てのプロセスは多くとも 1 度だけ deliver(m) を呼び出す

プロセスは、ブロードキャストされたメッセージを受け取ると、そのメッセージを正確に動作している全てのプロセスに届ける。このようにすることで 1 部分のプロセスだけがメッセージを受け取ることを防止している。図 3.4 は、上記の 3 つの特性を満たしたブロードキャストである。

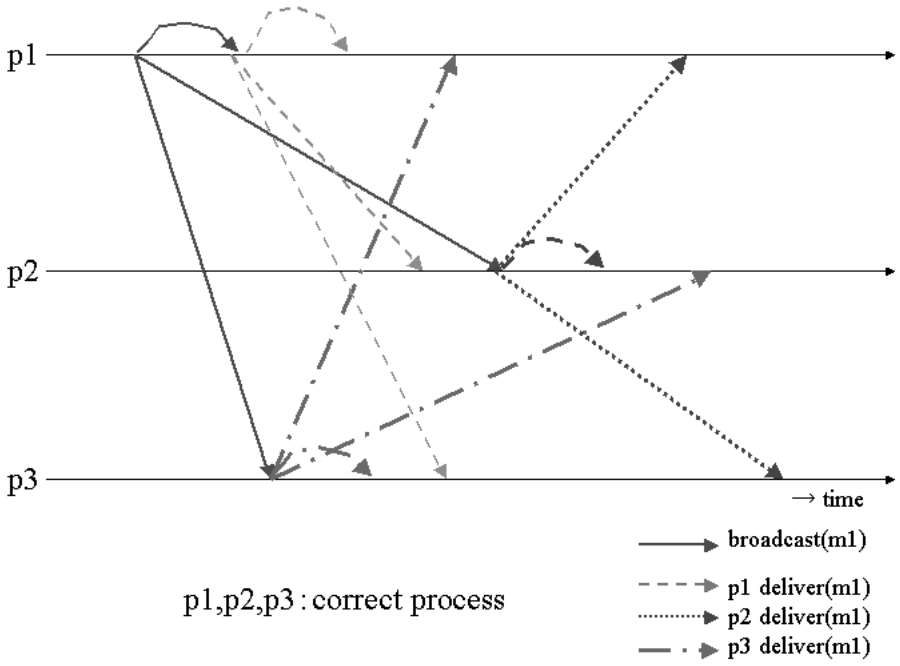

図 3.4: 有効性、合意性、整合性を満たしているブロードキャスト

3.2.5　故障が引き起こす可能性

次に Reliable broadcast がメッセージをブロードキャストしている際に、故障が発生した場合に引き起こす可能性について説明する。

1. 正確に動作している全てのプロセスは、メッセージを受け取る
2. どんなプロセスもメッセージを受け取らない

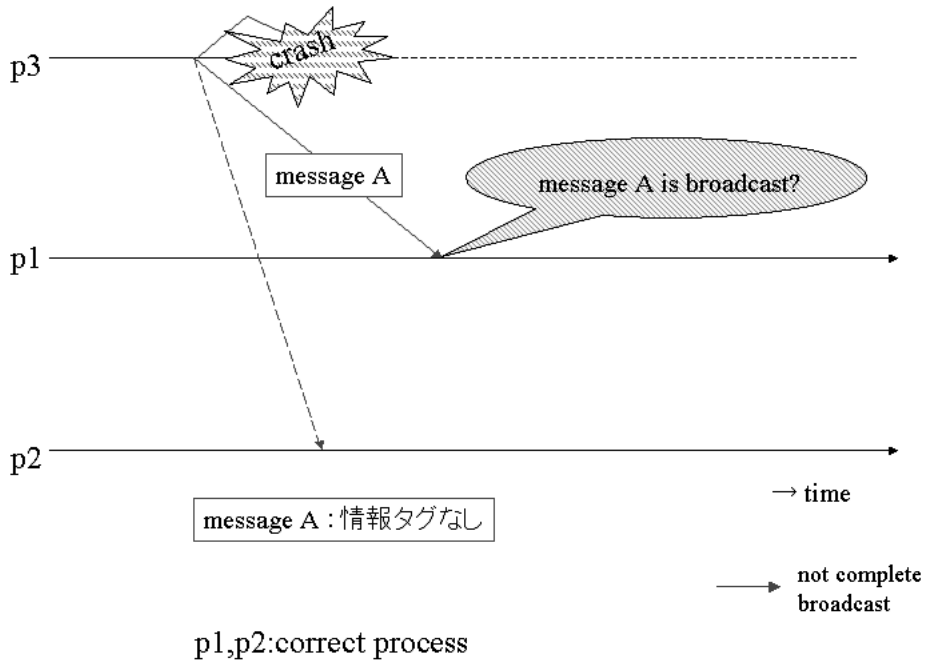

図 3.5: 故障が引き起こす可能性 2

まず、故障が引き起こす可能性 2 について図 3.5 を使い説明する。プロセス 3 がメッセージ A をブロードキャストしようとする時にプロセス 3 が故障したら、正確に動作しているプロセスは決して、メッセージ A を受け取ることはない。それは、プロセス 3 がメッセージ A をブロードキャストしようとしたことの分かる情報タグが付いていないからである。

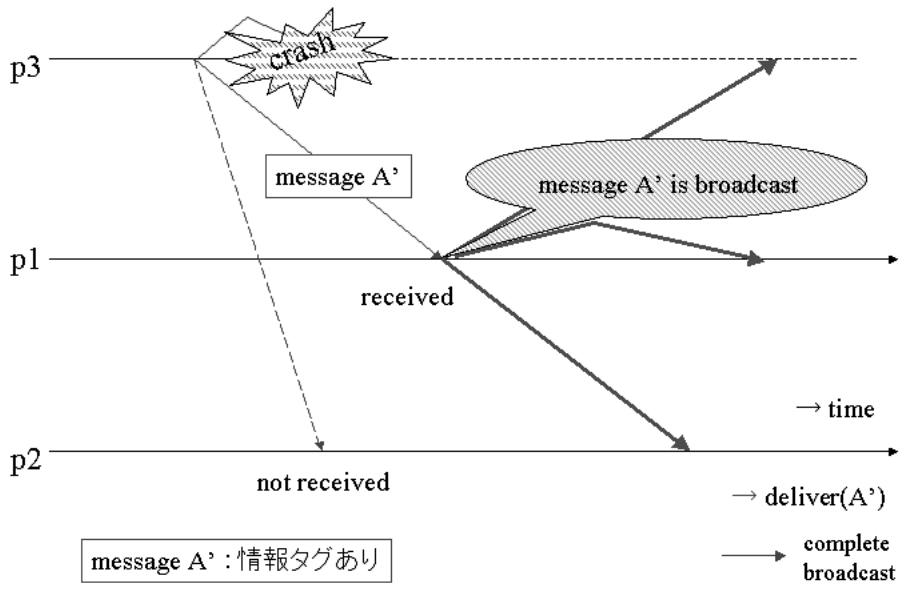

図 3.6: 故障が引き起こす可能性 1

次に故障が引き起こす可能性 1 について図 3.6 を使い説明する。プロセス 3 がメッセージ A' をブロードキャストしている最中にプロセス 3 が故障した。しかし、プロセス 3 がメッセージ A' をブロードキャストしたことが分かる十分な情報（メッセージの情報タグ）が送信された場合、正確に動作しているプロセス 1 は、このメッセージ A' を受け取ることができるかもしれない。その結果、正確に動作している全てのプロセスは、メッセージ A' を受け取ることができる。

3.2.6　Reliable broadcast と合意問題の関係

合意問題では、正確に動作しているプロセスが初期値を提案する際に、この初期値に関する情報メッセージが、合意処理に参加している全てのプロセスに確実に届く必要がある。１部分のプロセスだけが、初期値に関する情報メッセージを受け取るのであれば、合意処理は正確に終了しないのである。また、合意処理を行った結果、決定値が全てのプロセスに届く場合も、初期値を提案する場合と同じである。これらのことから、Reliable broadcast は、合意問題の中で、メッセージの送受信に関する大切な部分を担っている。

3.3　合意問題について

複製技術の実装をする際にプロセス集合の合意問題は、不可欠な部分となる。それは、分散システム上で複製サービスを提供するということは、複製サーバーの中の複製プロセス集合が矛盾のない状態を共有維持することが求められる。したがって、合意問題が必要なのである。合意とは、あるプロセス集合内で故障が発生しても、そのプロセス集合が共通の値（ここでの共通の値とは、プロセス集合のが合意のために提案した値に依存した値を指す）の決定にたどり着くことを可能にすることである。

3.3.1　合意問題による通信基本命令

合意問題は、次の２つの通信基本命令で定義されている。図 3.7 は、この２つの通信基本命令を使って合意処理の流れを簡単に示したものである

propose(v) ：値ｖを提案する

decide(v) ：値ｖに決定する

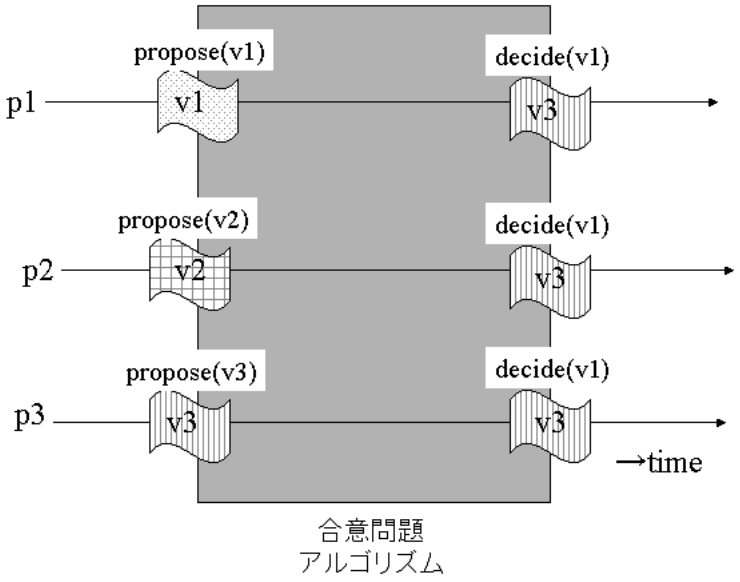

図 3.7: 合意問題の流れ

3.3.2　合意問題の特性

合意問題は、次の４つの特性を満足する必要がある。

Termination(有効性)
正確に動作している全てのプロセスは、最終的に任意の値に決定する

Validity(有効性)
もし、値を提案する全てのプロセスが propose(v) を呼び出すなら、正確に動作している全てのプロセスは最終的に decide(v) を呼び出す

Agreement(合意性)
正確に動作しているプロセスがある値に決定に下したなら、正確に動作している全てのプロセスはこの値に決定を下す

Integrity(整合性)
全てのプロセスは、多くても 1 度だけしか合意の決定を下さない。そして、全員が一致した値は、必ず提案された値でなければならい

有効性とは、提案された値が全て一致しているときは、その値に合意の決定を下さなければならないことを示している。もし、提案された値がバラバラで同じでないときは、整合性の特性によって、決定される値は、提案された値の中から選ばれることを保障している。

これから、説明する合意問題は、Chandra&Toueg の合意問題である。この合意問題は Unreliable Failure detector を使って合意問題を解決している。Semi-passive replication の基本となっている Lazy consensus は、この Chandra&Toueg の合意問題を変更したもである。

3.3.3 Chandra&Toueg の合意問題

Chandra&Toueg の合意アルゴリズムは、合意アルゴリズムを実行するプロセス集合の大半が正確に動作していることを前提にしている [5]。図 3.8 は、プロセスの故障がない場合の Chandra&Toueg の合意アルゴリズムによる合意処理の流れを示している。最初に図 3.8 で示されている用語について説明する。

プロセスについて

p1,p2,p3,p4,p5 :複製サーバーの中で動いている複製プロセス

p1 :コーディネータプロセス

プロセスが使うイベント

estimate :各々プロセスが持つ推定値をコーディネータプロセスに送っている

propose :提案値を全ての複製プロセスに提案する

ack :コーディネータプロセスからの提案値に賛成すること知らせている

decide :提案値が全ての複製プロセスに合意されて、決定値になったことを知らせている

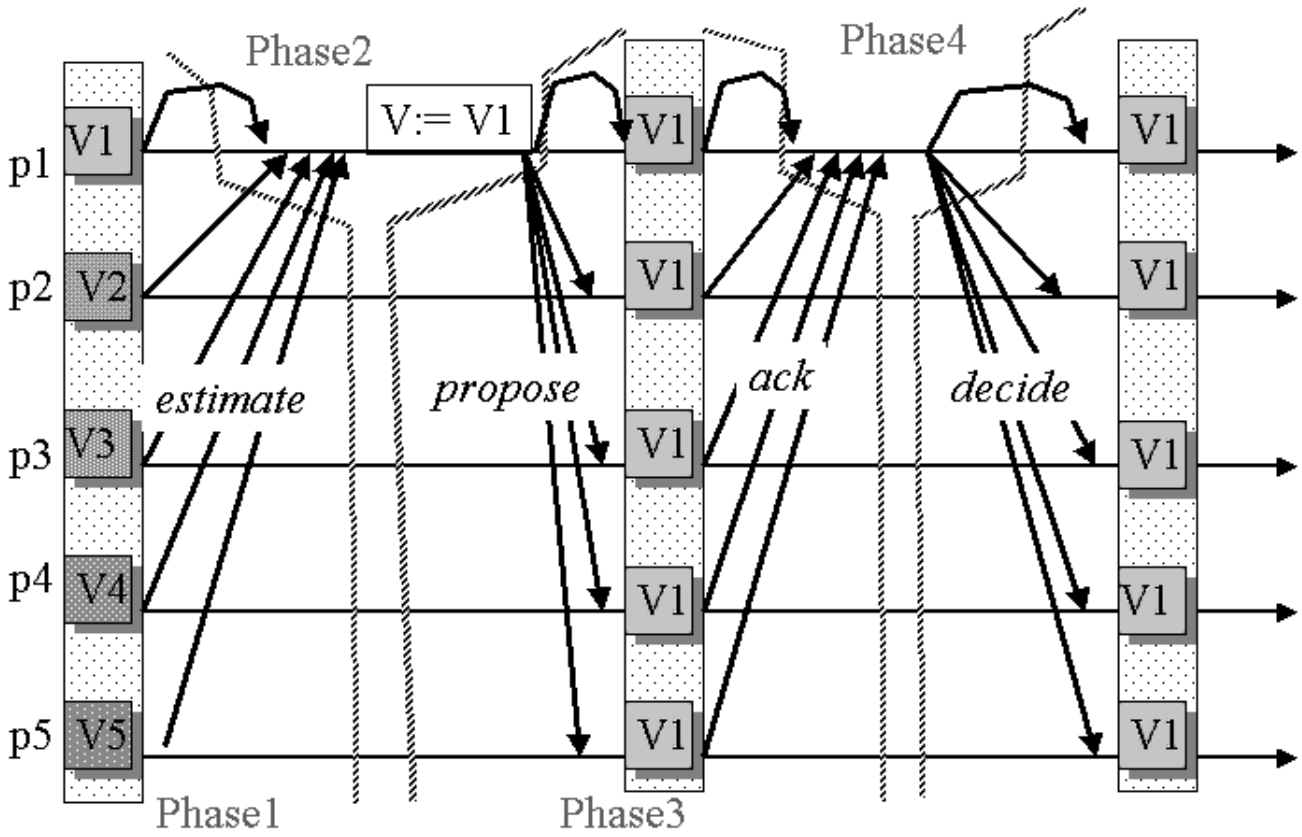

図 3.8: Chandra&Toueg の合意問題

3.3.4 Phase1-Phase4 について

Phase1

複製サーバーの中の全ての複製プロセスが、推定メッセージをコーディネータプロセスに送る。この時の推定メッセージには、推定値を一番最後に変更したラウンド番号がついている。

Phase2

コーディネータプロセスは、複製プロセス集合がコーディネータプロセスの送ってきた推定メッセージの中から、提案メッセージとともに送る提案値を手にいれる。コーディネータプロセスは、タイムスタンプの一番高い値のついている推定値を選ぶ。そして、この推定値を自分の推定値に書き換える。それから、複製サーバーの中で動いている全ての複製プロセスに提案メッセージをブロードキャストする。

Phase3

複製プロセス集合は、コーディネータプロセスからの提案メッセージが届くのを待っている。複製プロセス集合は、提案メッセージを受け取ると各々の複製プロセスが持っている推定値と推定値に付加するタイムスタンプを提案メッセージについてきた提案値とタイムスタンプに書き換える。そして、コーディネータプロセスに賛成応答メッセージを送る。複製プロセスが、コーディネータプロセスからの提案メッセージを受け取る前に、Failure detector によってコーディネータプロセスが故障したかもと疑われた場合、その複製プロセスは、コーディネータプロセスに拒否応答メッセージを送った後、次のラウンドに移る。

Phase4

コーディネータプロセスは、複製プロセス集合からの応答メッセージの大半が届くの待っている。この時の応答メッセージは賛成応答メッセージかもしれないし、拒否応答メッセージかもしれない。

もし、コーディネータプロセスの元に届いた全ての応答メッセージが賛成応答なら、Phase3 でコーディネータプロセスが提案メッセージに付加した提案値が全ての複製プロセスが合意を下した決定値となる。このとき、コーディネータプロセスは、決定メッセージに決定値を付加してブロードキャストする。これにより、１つの合意処理が終了したことになる。対照的にコーディネータプロセスの元に集まった応答メッセージの中に１つでも拒否応答メッセージがある場合には、コーディネータプロセスは合意処理をあきらめて、つぎのラウンドの新しい Phase に移る。

3.3.5　合意処理の中で重要となる値

ラウンド番号

各々の複製プロセスが合意処理を実行しているラウンド番号。このラウンド番号によってそれぞれのラウンドの唯一のコーディネータプロセスが決定する

推定値

推定値とは、合意処理が始まる前に各々の複製プロセスが計算処理して算出した値である。この値は、合意処理の中で提案されて、合意の決定を下されるであろう値である。つまり、推定値が提案値になり、決定値になるのである

タイムスタンプ

推定値に関するタイムスタンプである。上記で示したように推定値が、提案値、決定値と変わる。したがって、推定値が変化した時のラウンド番号がタイムスタンプである

3.3.6　コーディネータプロセスの決定

コーディネータプロセスは、ラウンド番号と複製サーバーの中で動いている複製プロセスの構成によって決定される。この複製プロセスの構成は、合意処理実行中にかわることはない。また、合意処理が始まる前に全ての複製プロセスに、複製プロセスの構成は知らされている。したがって、コーディネータプロセスの決定は、次の式によって意思決定的に行われる。

$$C^r = ((r-1) \bmod n) + 1$$

C :コーディネータプロセスとなるプロセス番号

r :ラウンド番号

n :複製サーバーの中で動く複製プロセスの数

複製サーバーの中で動く複製プロセスに、あらかじめ通し番号を付けておく。そして、この式によって算出された値と同じ番号のプロセスがコーディネータプロセスとなる。

コーディネータプロセスの決定例　複製プロセスの数が 5、ラウンド番号 3 の時のコーディネータプロセスは次のように算出される。

$$C^3 = ((3-1) \bmod 5) + 1 = 3$$

複製プロセスの数 :5(p1,p2,p3,p4,p5)

ラウンド番号 :3

したがって、プロセス３ (p3) がコーディネータプロセスとなる。

プロセスの故障がない場合、合意アルゴリズムは図（）のように 1 ラウンド（Phase1-Phase2) で終了する。Chandra&Toueg の合意アルゴリズムは、非同期なラウンドの中で実行されている。非同期なラウンドとは、複製プロセスの合意処理のスピードが決まっていないということ。つまり、各々の複製プロセスのラウンドはバラバラに進むということである。ラウンドには連続したラウンド番号が付いている。このラウンド番号によって、それぞれのラウンドは識別されている。そして、複製プロセスが使うプロトコルメッセージも複製プロセスが動いているラウンドのラウンド番号によって識別されている。また、ラウンドが非同期であるから、いくつかのラウンドは、実際にはシミュレーション的に行われる。しかし、連続したラウンド番号によって論理的に順序づけられている。

3.4 複製モデル

Active replication、Passive replication そして、本研究で実装する Semi-passive replication はクライアント-サーバー型の複製モデルである。図 3.9 は一般的な複製アルゴリズムによる複製サービスの流れを示している。この図を使って各々プロセスが実行するイベント、使われるメッセージについて説明する。

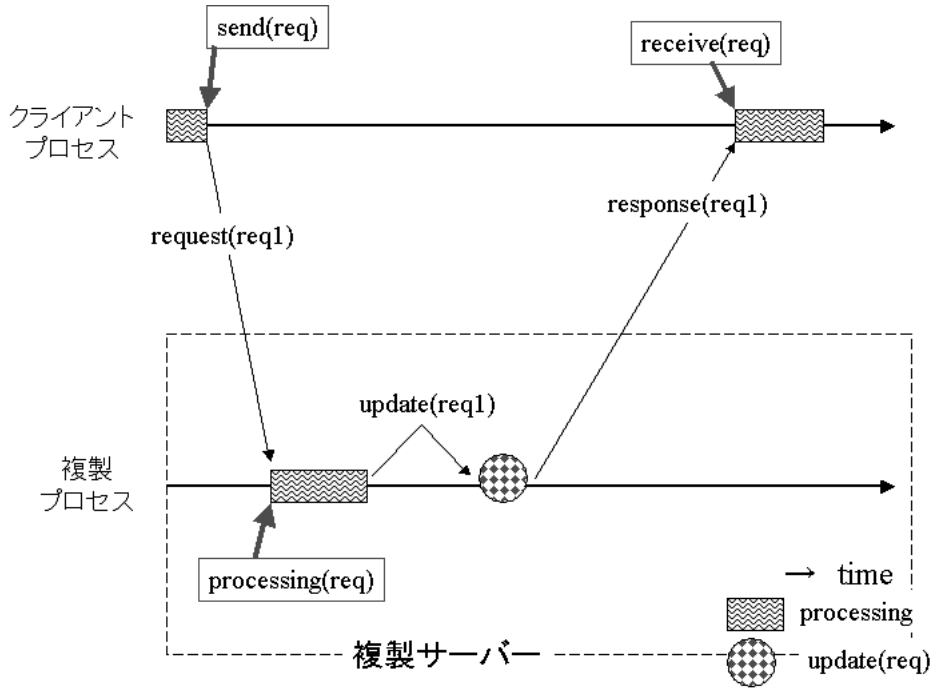

図 3.9: 一般的な複製技術による複製サービス

プロセスについて

クライアントプロセス 複製サービスを要求するプロセス

複製プロセス 複製サービスを実行するプロセス

クライアントプロセスが使うイベント

send(req) 複製サーバーに要求メッセージを送る

receive(resp) 複製サーバーからの応答メッセージを受けとる

複製プロセスが使うイベント

processing(req) クライアントプロセスからのサービス要求を処理する

update(req) 要求処理結果の更新、このイベントは意思決定的でなければならない

メッセージについて

request(req) 複製サービスの要求メッセージ

update(req) データ更新のためのメッセージ

response(req) クライアントプロセスへの応答メッセージ

3.4.1 複製技術の定義

複製技術の定義は、Active replication、Passivereplication の操作方針に関わらず共通の複製技術となる。そして、各々の複製技術の操作方針はこの複製技術の定義に特有の性質を追加することで定められる。以下に複製技術の定義を満足する４つの特性について示す。

- Termination（終了性）

もし、正確に動作しているクライアントプロセスが、複製サーバーにサービス要求を送るなら、最終的には、このクライアントプロセスは複製サーバーからの要求処理結果の応答を受け取る。

- Total order（イベントの全順位性）

クライアントプロセスからの複製サービス要求 *req1* と *req2* に対して、もし、任意の複製プロセスが *update(req1)* の後に *update(req2)* を実行したなら、ある複製プロセスは textslupdate(req1) の後に textsluodate(req2) を実行する。

- Upadat integrity（データ更新の整合性）

いかなる複製サービス要求 textslreq も、明らかにクライアントプロセスが複製サーバーに送った場合に限り、複製プロセスは多くても 1 度だけ *update(req)* を実行する。

- Respons integrity（応答の整合性）

クライアントプロセスに届けられた要求処理結果は、複製サーバーの中で正確に動作している複製プロセスによってデータ更新された要求処理結果である。

上記で示した４つの特性を満たすことで複製技術を実装した複製サーバーは正しく動作する。図 3.10 には４つの特性を全て満たした例を示し、図 3.11 には全順位性を満たしていない例を示す。

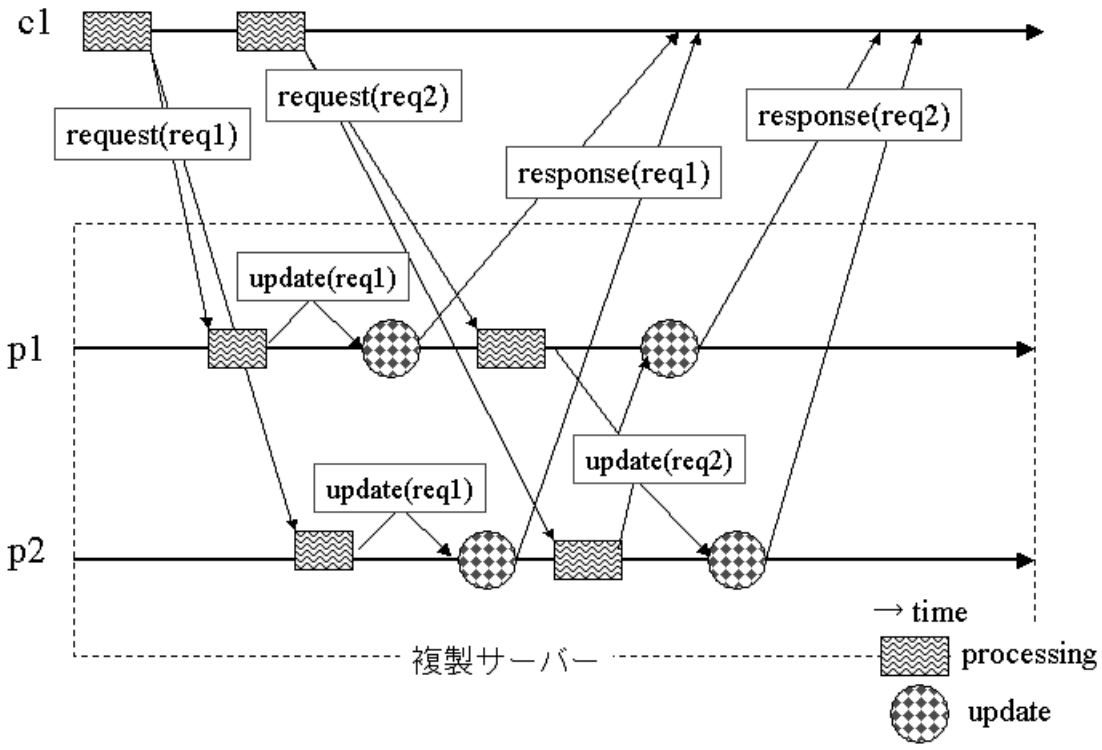

図 3.10: ４つの特性を満たしている複製サービス

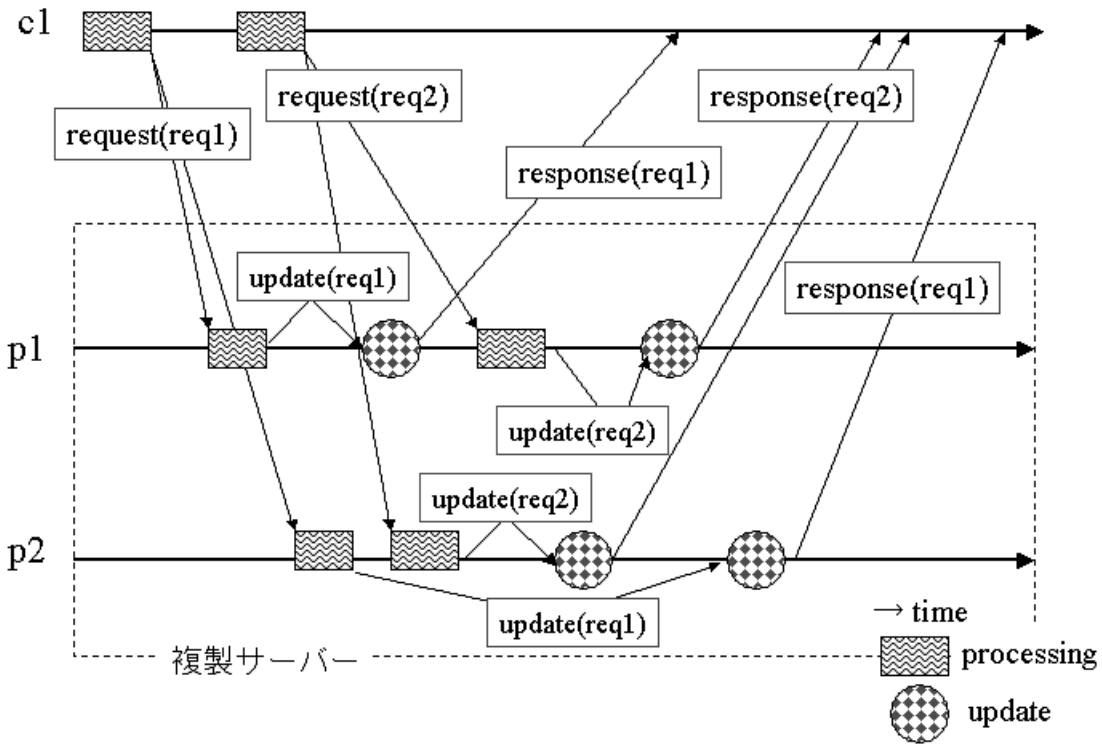

図 3.11: イベントの全順位性を満たしていない複製サービス

3.5 Passive replication

Passive replication は「primary-backup approach」とも呼ばれている [8]。Passive replication では、複製サーバーの中の 1 つのプロセスを集中制御プロセスとして使う。このプロセスのことを「primary プロセス」と呼ぶ。primary プロセスは複製サーバーの中で以下の役割を果たす。また、primary プロセス以外のプロセスは、primary プロセスからの update メッセージにともないデータ更新をする back プロセスとなる。図 3.12 は、Passive replication の階層図である。

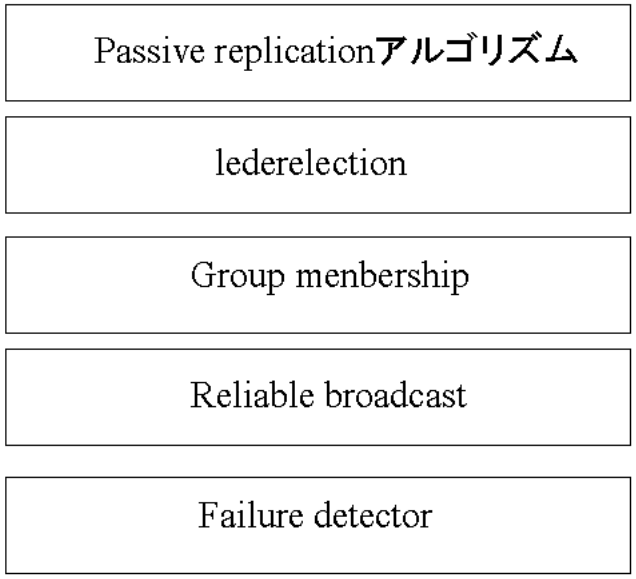

図 3.12: Passive replication の階層図

3.5.1 primary プロセスの役割

複製サーバーの中で primary プロセスだけがクライアントプロセスからの呼び出しメッセージを受け取る。そして、要求処理の結果をクライアントプロセスに送る。primary プロセス以外のプロセスは、backup プロセスとなる。backup プロセスは、primary プロセスとだけメッセージ交換を行う。もし、primary プロセスが故障したら backup プロセスの中から新しい primary プロセスが選ばれる。

3.5.2 Passive replication の複製サービス

次に Passive replicatioin の複製サービスの流れについて図 3.13 を使って説明する。

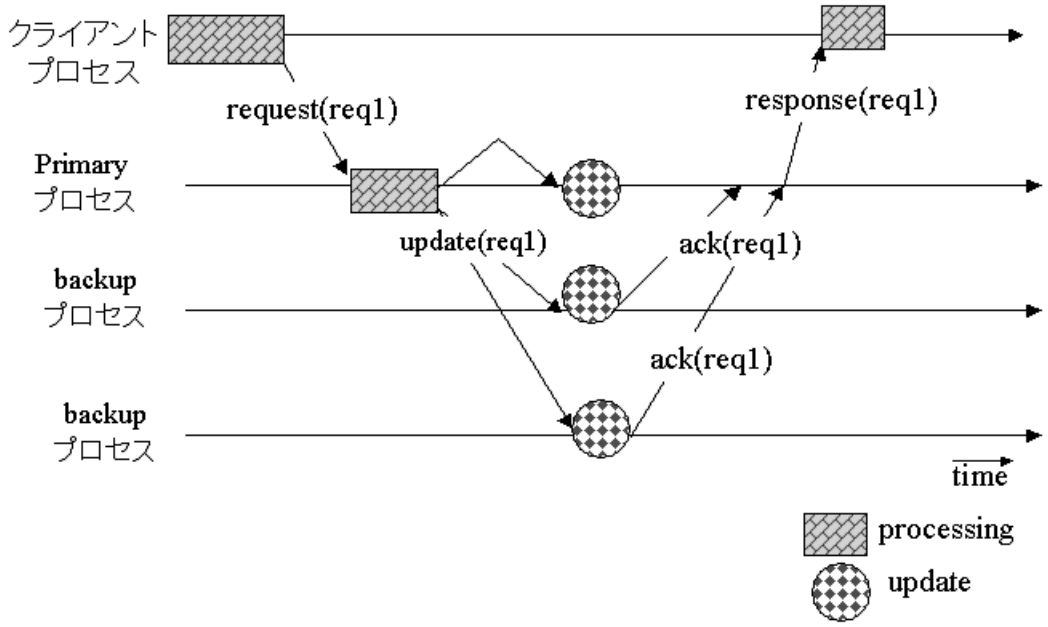

図 3.13: Passive replication の複製サービス

Passive replication の複製サービスの流れ

1. クライアントプロセスは、primary プロセスに唯１つの呼び出し識別子のついたメッセージを送る
2. primary プロセスは、クライアントプロセスからの呼び出しメッセージを受け取ると要求処理を行う。そして、自分自身と他の backup プロセスにデータ更新メッセージを送る。backup プロセスは、データ更新メッセージを受け取ると自分の状態を更新して、primary プロセスに賛成応答メッセージを送る
3. primary プロセスは、正確に動作している全ての backup プロセスからの賛成応答メッセージを受け取るとクライアントプロセスに応答メッセージを送る

3.5.3 Passive replication の利点と欠点

まず、複製プロセスの故障がない場合（図 3.13）について考える。Passive replication では、primary プロセスだけがクライアントプロセスからの要求処理を行うので、全ての複製プロセスがクライアントプロセスからの要求を処理している Active replication に比べてプロセスの計算処理コストが少なよい。次に backup プロセスは、primary プロセスからのデータ更新メッセージを受け取り、自分自身の状態を更新すればよい。したがって Passive replication を実装する際には状態の合意に関する層は必ずしも必要ではない。しかし、primary プロセスからのデータ更新メッセージの集合（メッセージの受取り順序も含む）は、一致する必要はあるので Reliable broadcast の層は必要である。

Passive replication の利点

a: Active replication に比べて複製プロセスの計算処理コストが少ない

b: 各々の複製プロセスの状態に関しての合意の層は必ずしも必要ではない

次に複製プロセスの故障がある場合について考える。Passive replication では、primary プロセスと backup プロセスの２種類のプロセスがある。backup プロセスは primary プロセスとメッセージ交換をするだけなので backup プロセスの故障はクライアントプロセスに影響はない。一方、primary プロセスが故障した場合、クライアントプロセスに影響を与える。それは、primary プロセスだけがクライアントプロセスとメッセージ交換をしていることと、複製サーバーの中で集中制御的な役割をしているからである。図 3.14 には、primary プロセスが故障するポイント（A,B,C）を示している。この図 3.14 を使って primary プロセスの故障がクライアントプロセスに与える影響を考えてみたい。

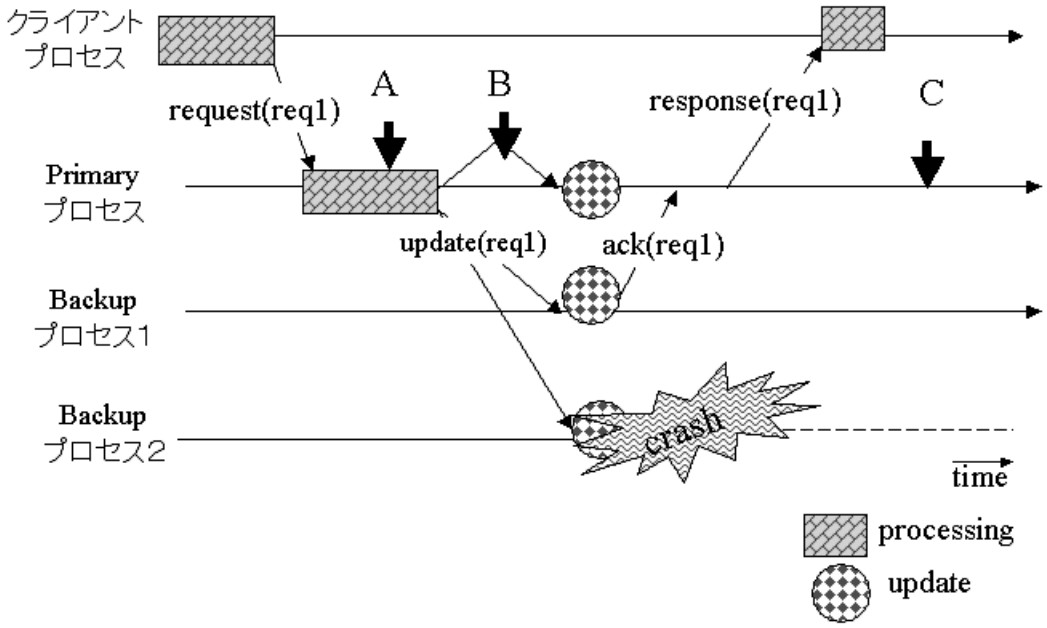

図 3.14: 故障が起きた場合の Passive replication の複製サービス

primary プロセスの故障が発生するポイントによって次のようにクライアントプロセスへの影響が変わってくる。

A: クライアントプロセスからの要求を処理している間に故障発生

B: backup プロセスにデータ更新メッセージを送った後に故障発生

C: クライアントプロセスに応答メッセージを送った後に故障発生

A,B,C の全ての場合において言えることは、新しい primary プロセスを選ぶ必要があるというと。次に各々の場合について考えてみる。

C の場合、クライアントプロセスが primary プロセスからの処理結果の応答メッセージを受け取った後に故障しているので、クライアントプロセスは Failure detector によって primary プロセスの故障が探知できて次の複製サービスを要求するときには、新しい primary プロセスに要求メッセージを送ることができる。したがって、primary プロセスの故障はクライアントプロセスにとって分かりやすい。

しかし、A と B の場合では、クライアントプロセスはいつま待っても primary プロセスからの応答メッセージを受け取ることができない。したがって再び、同じ要求を primary プロセスに送る必要があるかもしれない。この場合の primary プロセスの故障はクライアントプロセスにとって分かりにくい。

A の場合では、新しい primary プロセスは、クライアントプロセスから前回と同じ内容の要求メッセージをを新しい要求メッセージとして受け取り、処理を行う。

しかし、B の場合では primary プロセスは、backup プロセスにデータ更新メッセージを送った後に故障している。この時、backup プロセスの一部だけがデータ更新メッセージを受け取る状態があると、新しい primary プロセスは操作が困難になる。つまり、B の場合では、全ての backup プロセスは、primary プロセスからのデータ更新メッセージを受け取るか、受け取らないかの２つの状態のうちのどちらか１つの状態にする必要がある（Reliable broadcast の必要性）。そのようにしたら、新しい primary プロセスは、操作を簡単にできる。

例えば、全ての backup プロセスが primary プロセスからのデータ更新メッセージを受け取ってない場合、B の操作は、A の操作と同じものになる。

一方、全ての backup プロセスが primary プロセスからのデータ更新メッセージを受け取った場合、新しい primary プロセスは、前回と同じ内容の要求呼び出しに対して、２度も要求処理を行う必要はなく、要求メッセージを受け取ると、すぐにクライアントプロセスに応答メッセージを送ればよい。ここで大切なことは、クライアントプロセスからの要求メッセージに識別情報タグがついていることである。（識別情報タグについては、Reliable broadcast で説明している）

A,B,C のことから、Passive replication では、使用する Failure detector によって Passive replication の実装は左右される。また、システムモデルも非同期モデルだと Passive replication の実装は困難である。

Passive replication の欠点

a: primary プロセスの故障がクライアントプロセスに大きく影響する

b: primary プロセスが故障したら、新しい primary プロセスを選ぶ必要がある

3.6 Active replication

Active replication は「state machine approach」とも呼ばれている [9]。Active replication は、サーバーを複製することや複製サーバーとクライアントプロセスのやり取りを調整することで分散システムの耐故障性を実装する一般的な複製技術である。また、Active replication は複製管理プロトコルの理解、設計の手助けとなっている。

Active replication では、Passive replication で集中制御的な役割をしていた primary プロセスを必要としない。Active replication では、複製サーバーの中の全ての複製プロセスは、クライアントプロセスからの要求メッセージを受け取る。そして、各々で要求処理した後にクライアントプロセスに応答メッセージを送る。図 3.15 は、Active replication の階層図について示したものである。

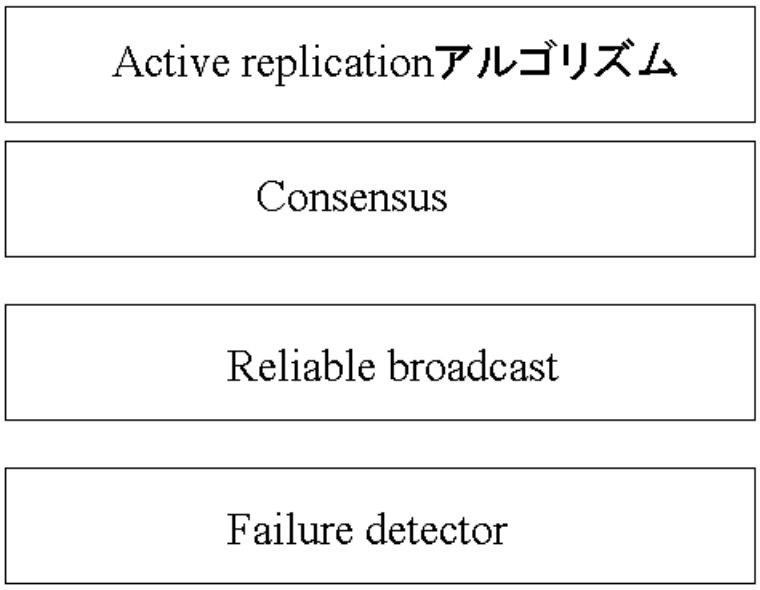

図 3.15: Active replication の階層図

3.6.1 Active replication の複製サービス

次に Active replication の複製サービスの流れについて図 3.16 を使って説明する。

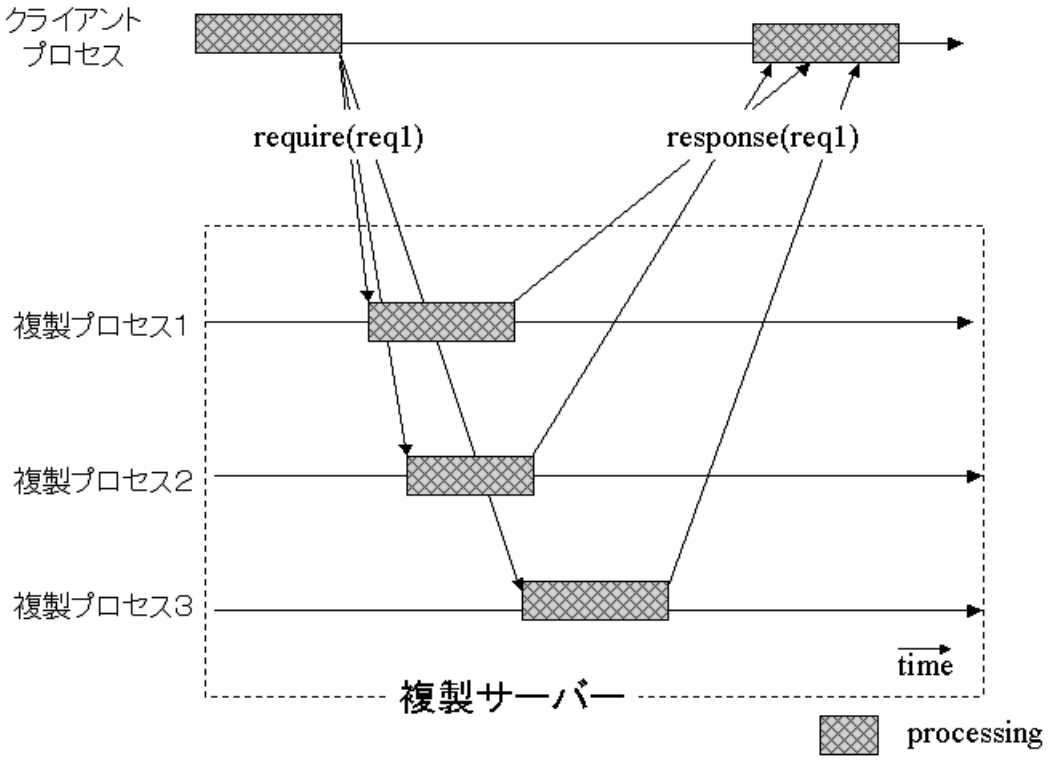

図 3.16: Active replication の複製サービス

Active replication の複製サービスの流れについて

1. クライアントプロセスからの要求メッセージは複製サーバーの中の全ての複製プロセスに送られる
2. 複製サーバーの各々の複製プロセスは、要求処理を行う。そして、クライアントプロセスに応答メッセージを送る
3. クライアントプロセスは、同一の応答メッセージの大半、または最初の応答メッセージを受け取るまで待機する。もし、複製プロセス集合が間違った操作をしないのであれば、クライアントプロセスは最初の応答メッセージを待つだけである。

3.6.2 Active replication の利点と欠点

図3.17 を使い複製プロセスに故障が発生した場合について考えてみる。Active replication では、全ての複製プロセス（Replica p1,p2,p3) がクライアントプロセス（Client) からの要求メッセージを受け取る。そして、各々の複製プロセスが要求処理をした後にクライアントプロセスに応答メッセージを送る。したがって、複製プロセスの故障は、クライアントプロセスに影響しない。また、クライアントプロセスは、複製プロセスの状態（複製プロセスが故障してるか、していないかという状態）を考える必要はないので、クライアントプロセスにとっては１つの複製プロセスに複製サービスを要求しているのと同じである。

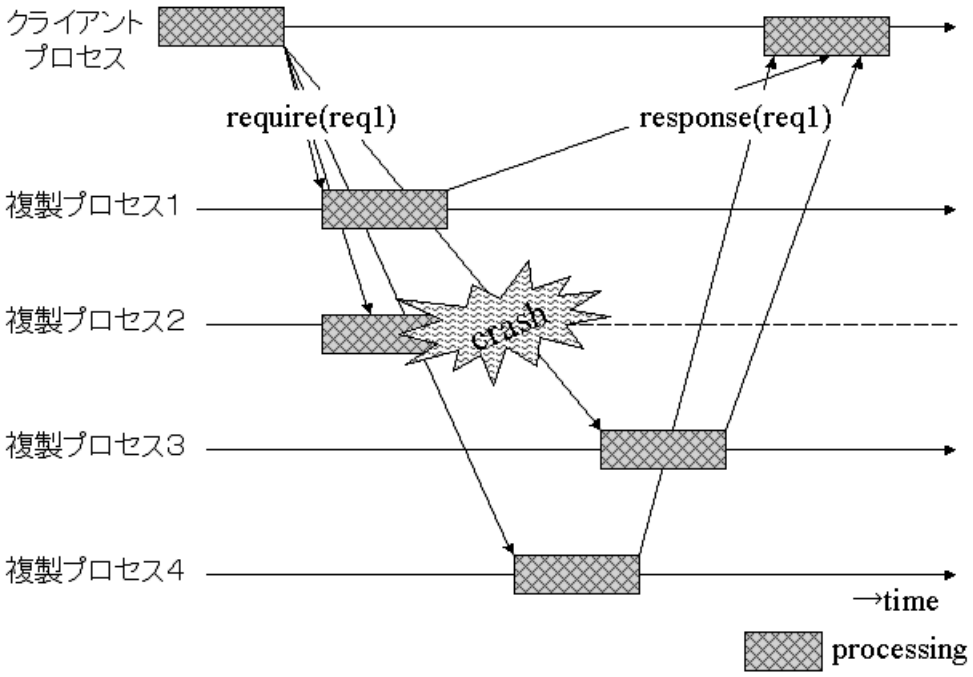

図 3.17: 故障が起きた場合の Active replication の複製サービス

Active replication の利点

a: クライアントプロセスに複製プロセスの影響がない

b: クライアントプロセスは複製プロセスの故障を考える必要がない

次に Active replication の欠点について考える。Active replication では、全ての複製プロセスが、クライアントプロセスからの要求に対して処理する。このことは、Passive replication で primary プロセスだけが要求処理しているプロセスの計算処理コストに比べて多くの計算処理コストを使う。また、各々の複製プロセスの状態は、同一である必要があるので複製プロセスの状態の合意問題が生じる。

Active replication の欠点

a: Passive replication に比べて計算処理コストが多い

b: 全ての複製プロセスの状態の合意が必要（合意問題の必要性）

3.7 Passive replication と Active replication の比較

Active replication では、全ての複製プロセスがクライアントプロセスからの要求を処理するので複製プロセスは、プロセスの最初の状態と入力情報によって唯１つの出力が決まる意思決定的であることが要求される。一方、Passive replication は、backup プロセスは primary プロセスからのデータ更新メッセージによって自分自身の状態を更新するので意思決定的でなくてもよい。

また、故障について Active replication は、クライアントプロセスに故障による待ち時間を与えない。そのうえ、クライアントプロセスは複製プロセスに故障が発生した場合の特別な処理は必要ない。しかし、複製プロセスの計算処理コストは大きい。一方、Passive replication は。複製プロセスの計算処理コストは少ない。しかし、複製プロセスに故障がある場合、特に primary プロセスが故障した場合にはクライアントプロセスに待ち時間を与える。そして、クライアントプロセスは故障に対する特別な操作が必要となる。

Active replication と Passive replication の利点の補完とは、複製プロセスの計算処理コストと故障によるクライアントプロセスへの影響のトレードオフである。Semi-passive replication では、Activereplication と Passive replication の利点を組み合わせて上記のトレードオフを解消している。しかし、Semi-passive replication の利点は、これだけではない。Semi-passive replication は、連続した要求処理に力を発揮する。これは、SEmi-passiive replication が Lazy consensus を基盤にしているからである。Lazy consensus については、９９９章の Semi-passive replication で説明する。

第4章　Semi-passive replication

4.1　Semi-passive replicationの概要

Semi-passive replication は、X.défago らによって考案された複製技術である。この複製技術の特色は、第３章で述べた Active replication と Passive replication の利点を組み合わせたことである。これら２つの複製技術の利点を組み合わせることによって、複製プロセスの計算処理コストが低く、そして、複製プロセスに故障が発生しても連続した複製サービスをに提供できるのである。Semi-passive replication では、Active replication と Passive replication の利点を組み合わせるために特別な合意問題を必要としている。図 4.1 は、Semi-passive replication の枠組みを表している。この図 4.1 の中の Lazy consensus という部分が Semi-passive replication の特色を支えているのである。

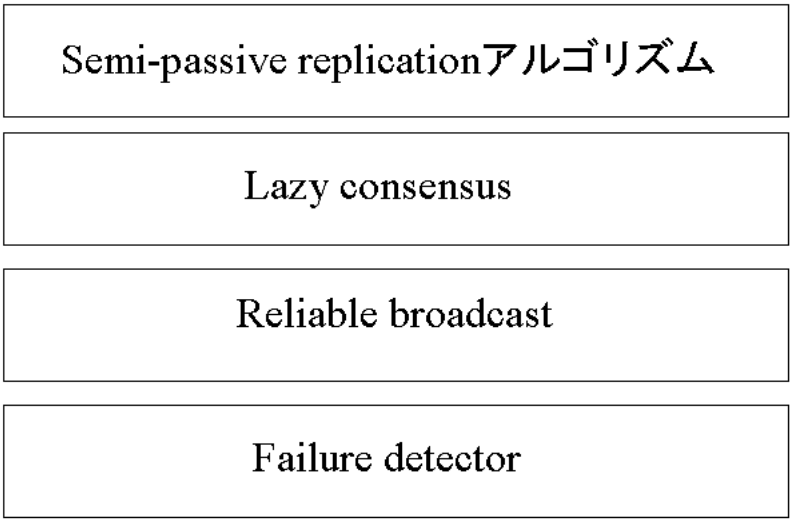

図 4.1: Semi-passive replication の枠組み

4.1.1 Semi-passive replication の複製モデル

Semi-passive replication は、クライアント-サーバー型の複製モデルとして定義している。また、複製サーバーの中の複製プロセスの大半は正確に動作していることを前提にしている。図 4.2 は、Semi-passive replication の中で使われるイベント、メッセージを示した Semi-passive replication の複製サービスである。

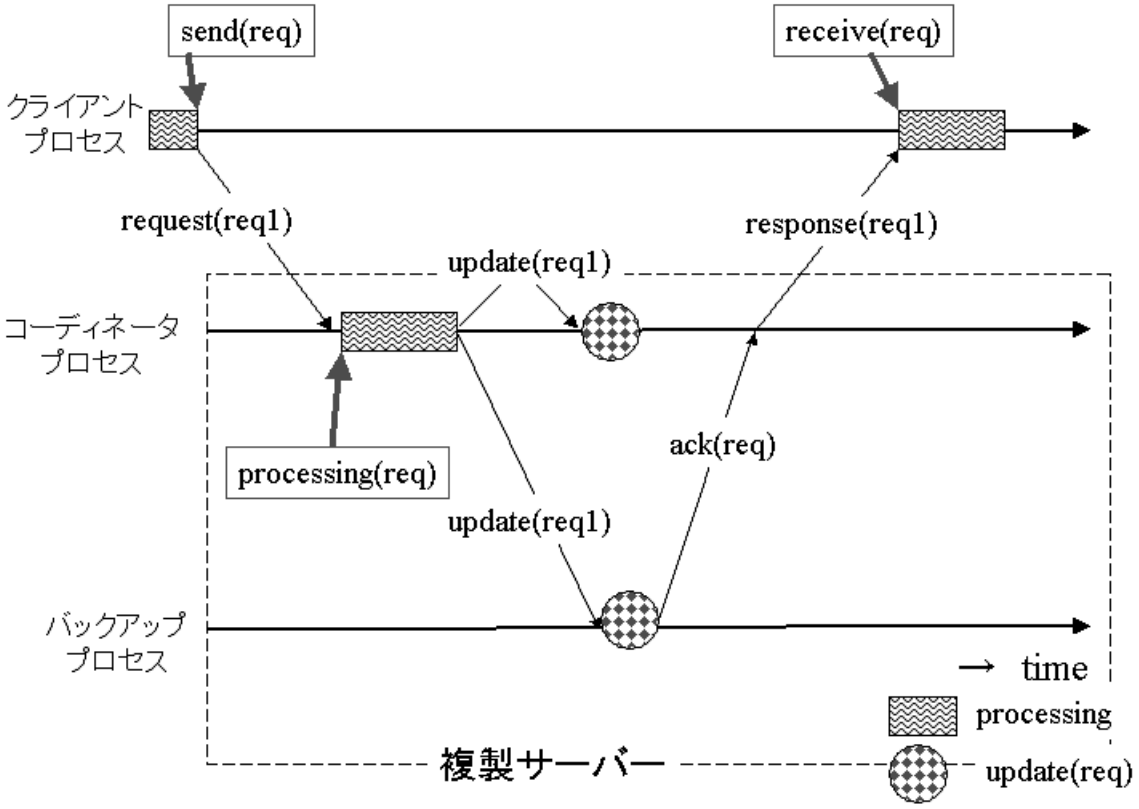

図 4.2: Semi-passive replication の複製サービス

プロセス

クライアントプロセス: 複製サービスを要求するプロセス

コーディネータープロセス: 複製サーバーの中で集中制御的役割をする

バックアッププロセス: コーディネータプロセスからのデータ更新メッセージによってデータ更新だけをするプロセス。しかし、コーディネータプロセスが故障した場合には、バックアッププロセスから新しいコーディネータプロセスが選ばれる

クライアントプロセスのイベント

send(req): 複製プロセスに複製サービス要求を送る

recive(req): 複製プロセスから複製サービスの応答を受け取る

複製プロセスのイベント

processing(req): 複製サービスの要求処理を実行する

update(req): データの更新をする。このイベントは意思決定的でなければならない

メッセージ

request(req): 複製サービス要求メッセージ

update(req): データ更新メッセージ

ack(req): データ更新完了応答メッセージ

nack(req):
送られてきたデータ更新メッセージを元にデータ更新することを拒否した応答メッセージ。この場合、Failure detector によってコーディネータープロセスが故障した可能性を知らされて、このコーディネータープロセスからのデータ更新メッセージを受け付けなかったことを表している

response(req): 複製サービス応答メッセージ

要求メッセージには、何番目の要求なのかを表すために req の後ろに番号がつく

4.1.2 Semi-passive replication の複製サービス

図 4.3 は、複製プロセスに故障が発生しない場合の Semi-passive replication の複製サービスを表したものである。以下に複製サービスの流れを説明する

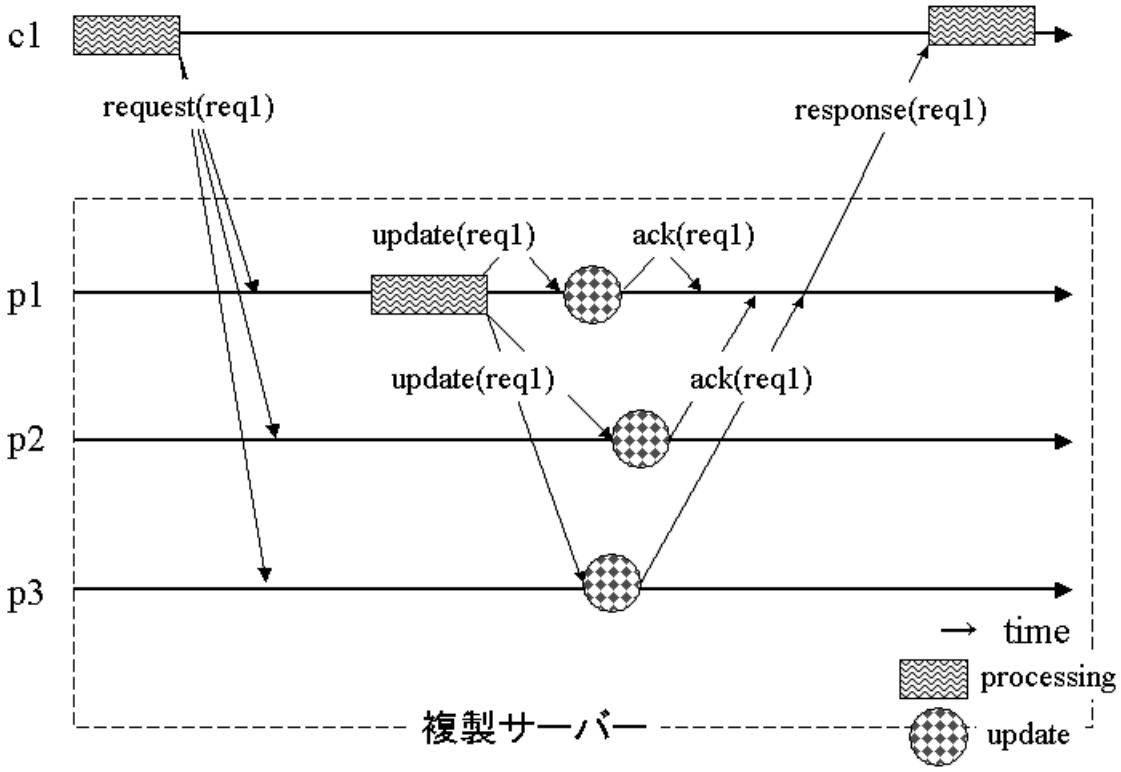

図 4.3: 故障がない場合の Semi-passive replication の複製サービス

複製プロセスに故障がない場合の複製サービス（図 4.3）

1. クライアントプロセス（c1）が send(req) を実行。複製プロセス (p1,p2,p3,p4) に複製サービス要求メッセージが送られる
2. 複製プロセスは、クライアントプロセスからの要求メッセージを受け取ると、コーディネータープロセス（p1）だけが要求処理を行う。そして、バックアッププロセス（p2,p3,p4）にデータ更新メッセージを送る
3. バックアッププロセスは、コーディネータープロセスからのデータ更新メッセージを受け取ると、update(req) を実行する。そして、コーディネータープロセスにデータ更新完了応答メッセージを送る

4. コーディネータープロセスは、大半のバックアッププロセスからのデータ更新完了応答メッセージが集まると、クライアントプロセスに複製サービス応答メッセージを送る
5. クライアントプロセスは、recive(req) を実行して、複製サービスの結果を受け取る

4.1.3 Semi-passive replication の特色

Semi-passive replication の特色は、複製プロセスの計算処理コストが少なく、クライアントプロセスに複製プロセスの故障の影響を与えないことである。このことを実現するために Semi-passive replication は、Passive replication と Active replication の利点を組み合わせている。図 4.4 は、Passive replication と Active replication の利点を強調したものである。

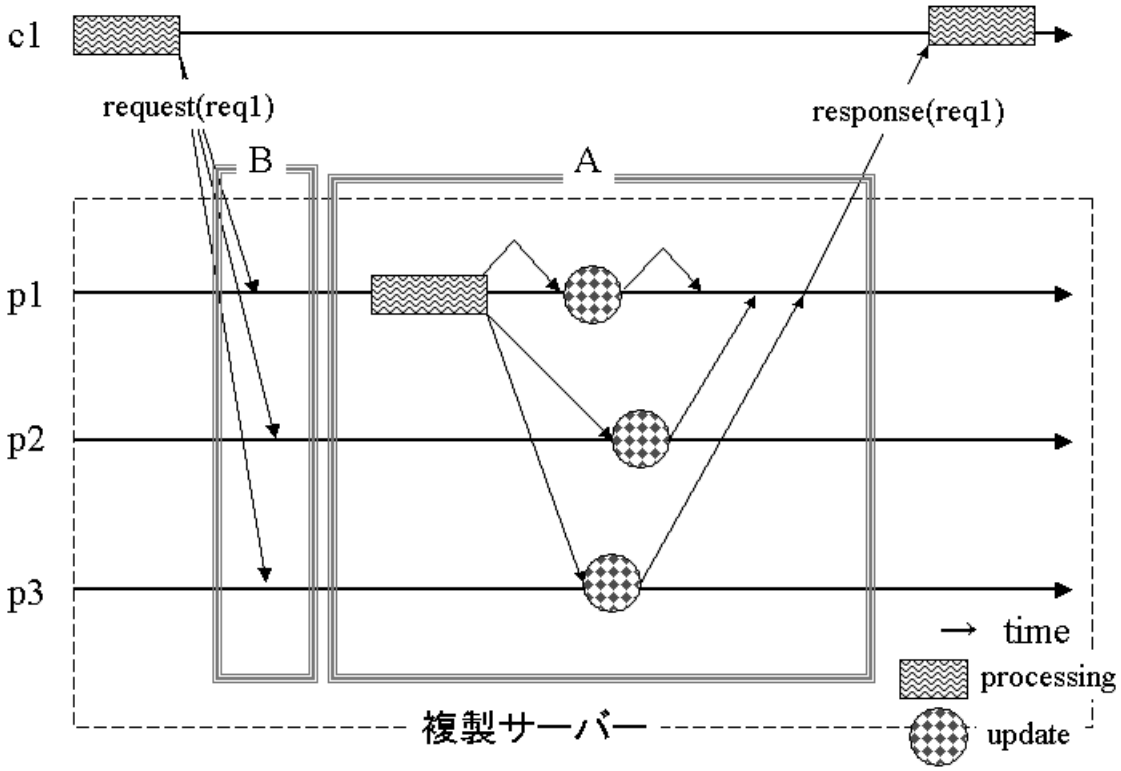

図 4.4: Active replication&Passive replication の特性を強調した Semi-passive replication の複製サービス

A: Active replication の利点

B: Passive replication の利点

以下に Passive replication と Active replication の利点を示す。

Passive replication の利点

複製サーバーの中で primary プロセスと呼ばれるプロセスが集中制御を行うので、プロセスの計算コストが少ない

Active replication の利点

複製サーバーの中の複製プロセスが故障しても複製サービスを必要としているクライアントプロセスに故障の影響を与えない

Passive replication の特性を活かしている部分　Semi-passive replication は、複製プロセスの中からコーディネータプロセスを選ぶ（図 4.4 では、p1 がコーディネータープロセス）。このコーディネータープロセスは、Passive replication の primary プロセスと同じ集中制御的役割をする。

このコーディネータープロセスのおかげで複製プロセスの計算処理コストは少なくてすむ。つまり、この部分が Passive replication の特性を活かしている（図 refactANDpassi の A）。

Active replication の特性を活かしている部分　Active replication では、複製サーバーの中の全ての複製プロセスがクライアントプロセスからの要求メッセージを受け取る。そして、各々の複製プロセスが要求処理を行い、クライアントプロセスに要求応答メッセージを送る。だから、クライアントプロセスに複製プロセスの故障の影響を与えないのである。

Semi-passive replication は、全ての複製プロセスがクライアントプロセスからの要求メッセージを受け取る。しかし、コーディネータープロセスだけが要求処理を行う。もし、コーディネータープロセスが故障しても、他の複製プロセス（図 4.4 の p2,p3）は、クライアントプロセスからの要求メッセージを受け取っているので、他の複製プロセスから選ばれる新しいコーディネータープロセスが、直ちに要求処理を行う。クライアントプロセスは、再度同じ内容の要求メッセージを送る必要はない。したがって複製プロセスの故障は、クライアントプロセスに影響しないのである。

つまり、全ての複製プロセスがクライアントプロセスからの要求メッセージを受け取ることが Active replication の特性を活かしている部分である（図 4.4 の B）。

4.2 Semi-passive replication を支えるもの

Semi-passive replication を支えているものは、Lazy consensus である。Lazy consensus とは一般的な Consensus に変更を加えた合意問題アルゴリズムである。図 4.5 は、Lazy consensus の合意処理をを分かりやすく表したものである。この図 4.5 から分かるように Lazy consensus と Consensus の違いを簡単に述べると、Lazy consensus は、合意処理に入る前に全てのプロセスが提案するための値（以下、初期値と呼ぶ）を計算する必要がない合意問題である。

start ：Lazy consensus を開始する

propose(v) ：値 v を提案する

decide(v) ：値 v に決定する

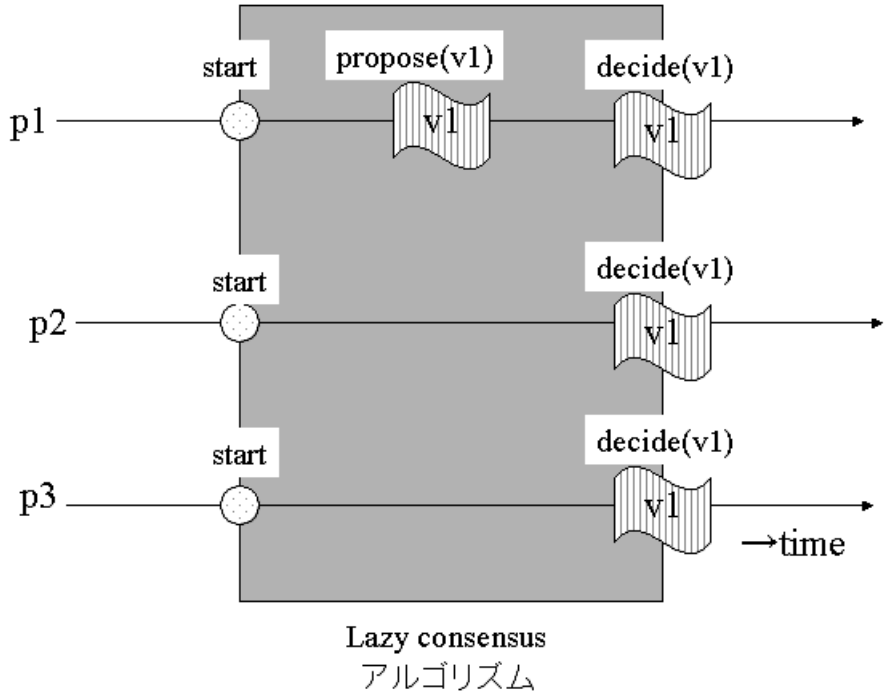

図 4.5: Lazy consensus による合意処理

ここでは、最初に Semi-passive replication の複製サービスが行われている間にプロセスの故障が発生した場合（図 4.6）を使って、Lazy consensus の利点を説明する。次に Lazy consensus と Consensus の違いを詳しく述べる。

4.2.1 Lazy consensus の利点

図 4.6 は、コーディネータプロセスが故障したときの Semi-passive replication の複製サービスを表している。図 4.6 では、次の新しいメッセージが使われる。

suspect(p): Failure detector によって故障した可能性のあるプロセスを知らせるメッセージ。Failure detector は、任意のプロセス間でメッセージを定期的に送信する。このメッセージがタイムアウトの間に届かない場合は、Failure detector を実装しているプロセスに suspect(p) メッセージを送る。p は、故障した可能性のあるプロセスを表す

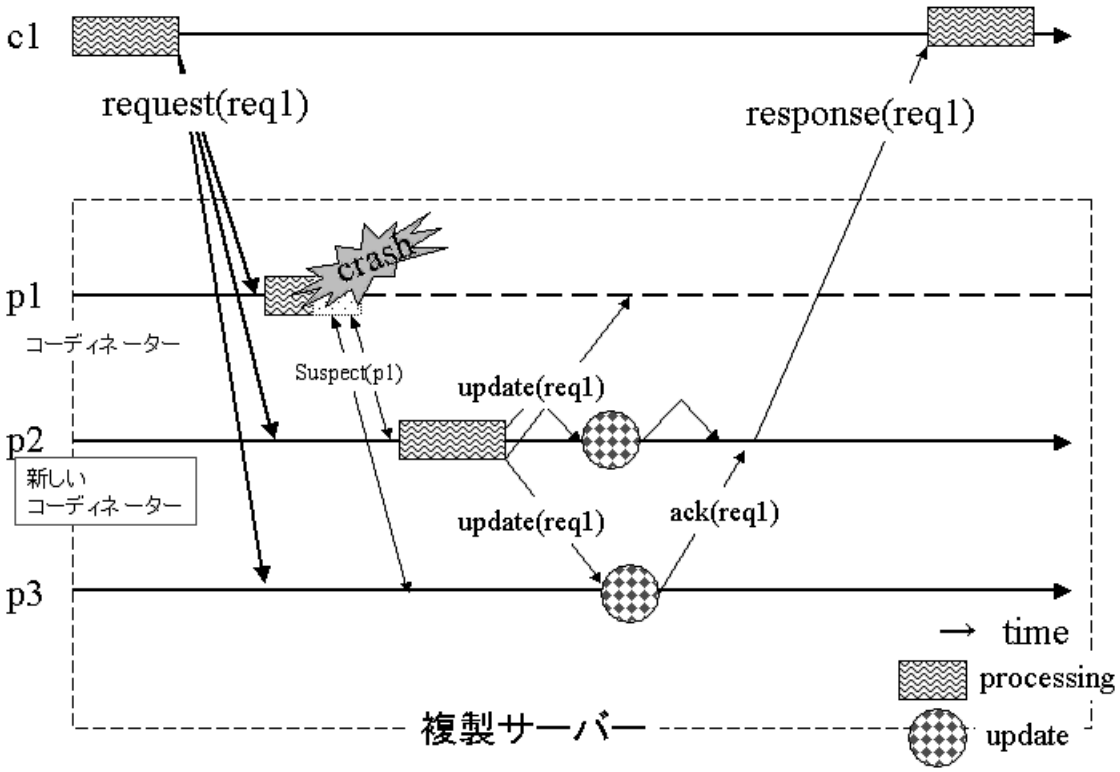

図 4.6: 故障が起きた場合の Semi-passive replication の複製サービス

故障が起きた場合の Semi-passive replication の複製サービス

1,2. プロセスによるイベント、イベントで使われるメッセージは、複製プロセスの故障がない場合（図 4.3）と同じである
3. backup プロセス (p2,p3) は、Failure detector の suspect(p1) メッセージによりコーディネータープロセス (p1) が故障した可能性を知る。backup プロセスたちは、新しいコーディネータープロセスを自分たちの中から選びだす。そして、新しいコーディネータープロセス (p2) によって、複製サービスを続行される
4. p2 は、backup プロセスにデータ更新メッセージを送る。このとき、p2 は故障した可能性のある p1 にもデータ更新メッセージを送る。p1 の故障は、可能性であり、もしかしたら、Failure detetor がミスをして故障したと判断したかもしれない。したがって、p2 は p1 にもデータ更新メッセージを送るのである
5,6. プロセスによって発生するイベント、イベントで使われるメッセージは、複製プロセスの故障がない場合（図 4.3）と同じである

Semi-passive replication は、全ての複製プロセス（図 4.6 の p1,p2,p3）がクライアントプロセスからの要求メッセージを受け取る。しかし、Active replication とは異なりコーディネータープロセスだけが要求処理をする。そして、要求処理結果の合意ををデータ更新イベントという形で行っている。もし、コーディネータプロセスが故障したら、次の新しいコーディネータープロセスだけが要求処理を行い、データ更新イベントを行う。したがって、Lazy consensus の利点を使っている部分は、コーディネータープロセスだけがクライアントプロセスからの要求を処理し、その要求処理結果をデータ更新イベントにより、全ての複製プロセスの状態を同じ状態している部分である。

表 4.1: Lazy consensus と Consensus(Chandra&Toueg) の比較

	プロセス数	タイミング
Lazy consensus	1	初期値が必要なとき
Consensus	全て	合意実行開始のとき

4.2.2 Lazy consensus と Consensus の違い

Chandra&Toueg の合意問題では、全てのプロセスが合意処理が始まるときに提案するための値（以下、初期値と呼ぶ）を計算する。これは、プロセスの計算処理コストを多く消費する。一方、Lazy consensus では、プロセスの計算処理コストを低くするためにコーディネータープロセスだけが初期値を計算する。次に、故障が発生した場合の２つの合意問題の操作について考える。Chandra&Toueg の合意問題では、コーディネーターリストにより、合意処理の各ラウンドでの唯一のコーディネーターを決定している。しかし、このコーディネーターリストのプロセスの構成順序は、いつも同じである。もし、連続した合意処理が発生した場合に、コーディネータリストの１番目にあるプロセスが故障していると仮定する。毎回の合意処理において、ラウンド２以降でしか、合意処理は終了しない。つまり、最初ののラウンドは無駄なのである。この無駄を省くために Lazy consensus では、毎回の合意実行のラウンド１で合意問題を終了させるために、値の合意処理を行うさいに、次の合意実行のためのコーディネータリストの合意も行っている（以下、このコーディネータリストのことを PV と呼ぶ）。[1]

[1]PV とは、Permutation Vector の省略である。

表 4.1 の中の、プロセス数の部分で 1 つとは、コーディネータープロセスだけを指し、全てとは、合意問題に参加する全てのプロセスを指す。

4.3 Neko について

Neko は、P.Urbán らによって開発された分散アルゴリズム実装のためのプラットホームである。Neko は次の場所で手にいれることができる。

http://lsrwww.epfl.ch/neko/

図 4.7 は、Neko の環境を簡単に表したものである。図 4.7 を見ると分かるように Neko は、実装した分散アルゴリズムを使ってシミュレーション実行、実ネットワーク実行の 2 つの評価環境で実装した分散アルゴリズムの性能評価が行うことができる。

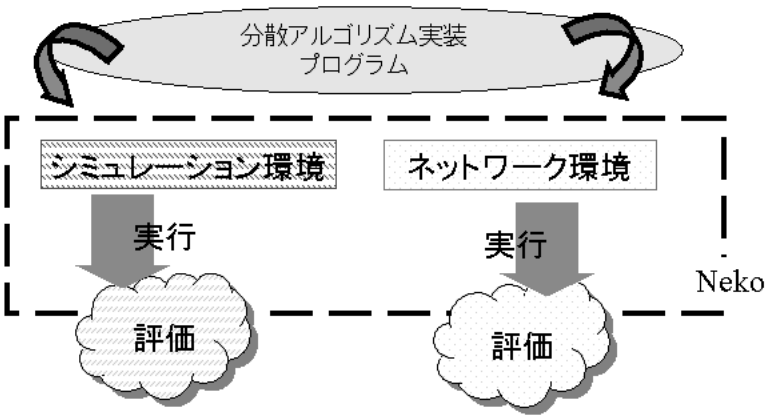

図 4.7: Neko について

また、Neko の中には、分散アルゴリズムを実装するときに役に立つ豊富コンポーネントが揃っている。第 3 章で説明した複製技術を実装するときに必要な合意問題に関するコンポーネントや、Faliure detector, Reliable broadcast なども揃っている。これらの、メリットは一人で分散アルゴリズムを実装するときには、時間もしくは、労力を軽減することができる。そして、Neko では、実装の途中段階の分散アルゴリズムをシミュレーショ

ン実行できるので、もし、分散あリゴリズムを実装しているプログラムに間違いがある場合には、すぐに変更できる。今回、この Neko を使って複製技術を実装したことで、分散アルゴリズムを実装するときに何が大切なのかということを経験した。それは、ただ、本で複製技術の概念を理解しているときには、わからなかったことである。Semi-passive replication のコンポーネント実装を通して感じたことは、Neko は、分散アルゴリズムを実装するには、大変便利なプラットホームである。それは、単に分散アルゴリズムを実装するときに役に立つコンポーネントが豊富にあるだけではない。Neko の中にあるコンポーネント実装した java プログラムを読むことで、アルゴリズムと実装プログラムの違いが発見できる。この発見したことを通して、アルゴリズムとプログラムのギャップを埋めたプログラムを勉強できる。また、neko のシミュレーション機能が、プログラム開発環境においてはもっとも役に立ったと思う。

第5章　実装について

Semi-passive replication の複製サービス（図 4.3）から考えると、Passive replication の枠組を Semi-passive replication の枠組に変更してから、Semi-passive replication の実装をした方が良いと思う。しかし、Passive replication は、非同期システムでは保障されていない。一方、Active replication は、非同期システムで保障されている。また、Neko の中には、図 5.1 の Active replication の枠組がある。本研究では、この Active replication の枠組を使って Semi-passive replivcation の実装をすることにした。Actve replication と Semi-passive replication の間で共通な部分は、特にプログラムの変更はしなかった。しかし、Lazy consensus アルゴリズムの階層で、Failure detetctor 層からのプロセスの故障可能性情報をもとに、コーディネータプロセス及びバックアッププロセスの操作を考える必要があるので、Failure detector 層と Reliable broadcast 層ではそれぞれの概念理解とプログラム動作確認のために多くの時間を費やした。

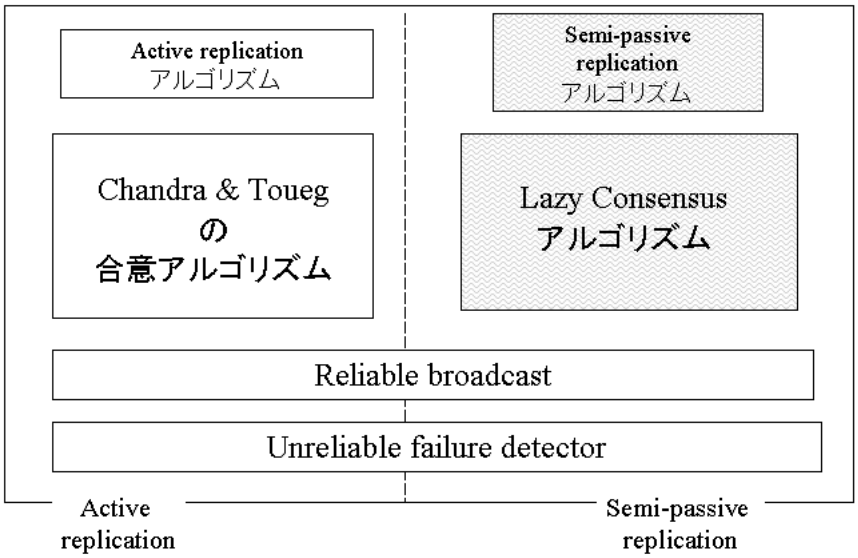

図 5.1: Neko の中にある Active replication の枠組

5.1 階層同士での通信について

Neko では、neko メッセージを使って異なる階層間での通信を行っている。このメッセージの中にそれぞれの階層特有の情報を付加して、異なるプロセス間の操作を行う。最初に Neko の使い方の習得と、分散アルゴリズムの基本となる異なる階層間でのメッセージ交換について学ぶために Reliable broadcast を使い２つのプロセス間でのメッセージ交換プログラムや複数のプロセスのメッセージ交換プログラム（図 refcircle）を作った。

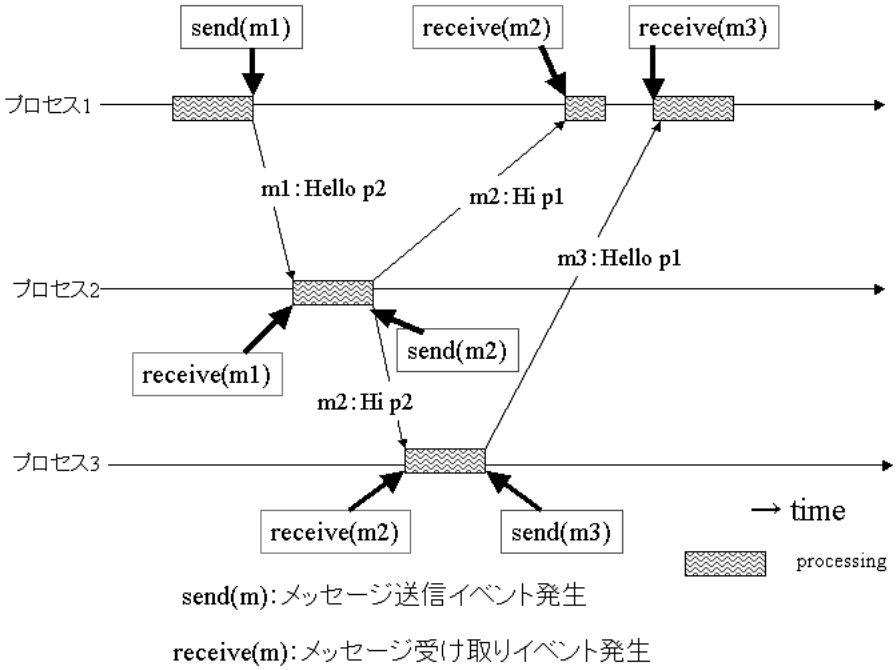

図 5.2: 複数のプロセスによるメッセージ交換の例

次に Failure detector の役割と、ハートビートメッセージのタイムアウトについて調べた。まず、故障発生プログラムを作成、このプログラムの実行結果が記録してある log ファイルを調べた。この時、分かったことが Neko のシミュレーションでは、１つのプロセスの中で、複数のイベントを同時に処理できないことである。また、合意処理に関する層の下に Heartbeat Failure detector に関する層を付け足して合意処理をするときには、プロセスの数とタイムアウトの時間調整や連続した合意処理の回数とタイムアウトの時間調整を最適にしないと、合意処理が終了しないことがわかった。ハートビートメッセージのタイムアウトについて、第 3 章複製技術の Failure detector で説明しているが、実際のプログラムを動かすことでタイムアウトがプロセスに与える影響が分かった。

ブロードキャスト
CheapRBcast.java
FakeRBCast.java
RBroadcast.java
Failure detector
FailureDetector.java
Heartbeat.java
OmegaFailureDetector.java
SimulatedFailureDetector.java
RingHeartbeat.java
....

表 5.1: ブロードキャストと Failure detector のコンポーネント

Neko の中には、ブロードキャストや Failure detector に関するコンポーネントが豊富にある（表 5.1）。ここでは、これらのコンポーネントについての説明はしない。しかし、着目して欲しいのは、Failure detctor のコンポーネントの多さである。このことは、分散システムの中で Failure detector がどれくらい大切かいうことがわかる。Lazy consensus のコンポーネントには、RBroadcast プログラムと Heartbeat プログラムを使用した。

5.2 Lazy consesus

まず、はじめに Neko の中にある Chandra&Toueg の ♢S Failure detector を使った合意アルゴリズムの実装プログラムをチェックした。合意アルゴリズムでは、１つのプロセス（コーディネータ）の合意処理の流れを記述している。しかし、アルゴリズム実装プログラムでは、バックアッププロセスの処理も考える必要がある。特にそれぞれのプロセスのラウンド処理には、２種類のプロセスのためのラウンド処理とプロセス故障発生のときのラウンド処理が大切なことが分かった。

5.2.1 初期値を持たない合意プログラム

図 5.3 は、シーケンス図である。この図 5.3 の start と get initial value の部分を Neko の中にある Chandra&Toueg の合意アルゴリズム実装プログラムを使った Lazy consensus プログラムに付け足した。get iniitial value の部分では、初期値を計算する giv() メソッドと、初期値があるか、ないかをチェックする check(value) メソッドを実装した。

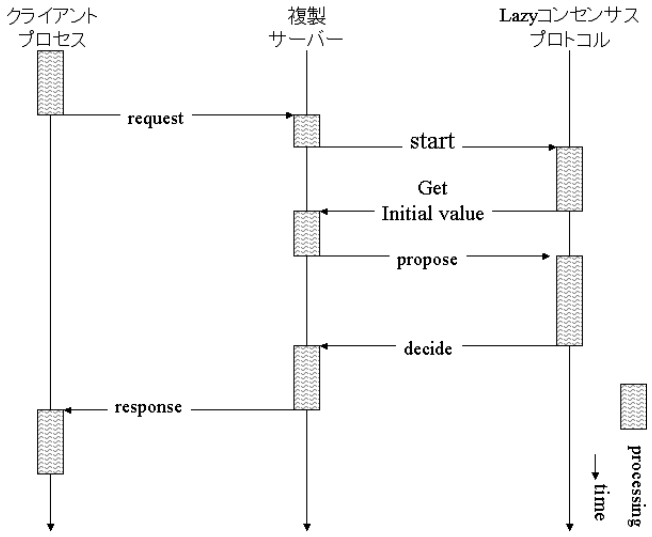

図 5.3: 複製サービスのシーケンス図

この時の Lazy consensus プログラムは、単に合意アルゴリズム実装プログラムを継承しただけなので、このプログラムの中身はないのと等しいものである。このように他のプログラム機能をそのまま、自分のプログラムに利用できるのは、１つは java のメリットである。そして、もう１つは、Neko の中にある豊富なコンポーネントが java で記述されていることである。

5.2.2 PV 処理の追加

最後に、Chandra&Toueg の合意アルゴリズム実装プログラムの機能を使わない Lazy consensus プログラムを作った。そして、この作ったプログラムに PV の合意処理を付け足して、連続した合意処理があるときには、PV の情報によってコーディネータが決定するようにした。このときに行ったことは次のとおりである。

- Neko メッセージのスタイル変更
- 連続した合意処理に適応した PV の処理
- Chandra&Toueg の合意プログラムと Lazy consensus プログラムの比較

最初の Neko メッセージのスタイル変更では、Chandra&Toueg の合意プログラムで使用されている Neko メッセージに PV の情報を追加した。次に、この Neko メッセージを使って決定値と PV の合意ができるように合意処理の部分を変更した。連続した合意処理に適応した PV の処理では、合意処理で決定された次の合意実行のための PV の情報を、静的

変数配列に記憶させた。そして、もし、次の合意処理があるときには、この静的変数配列の情報を使うことにした。

最後の Chandra&Toueg の合意プログラムと Lazy consensus プログラムの比較では、次の 3 ケースのプロセスの故障のシナリオを使って比較した。

1. 値を提案する前のプロセスの故障
2. 値を提案した後のプロセスの故障
3. ランダムなプロセスの故障

故障発生ケース１と２では、合意実行 1 回目の最初のラウンドのコーディネータプロセスの故障である。故障発生ケース３では、故障するタイミング、故障するプロセスもランダムである。このとき、プログラムの比較テストをしたことで合意処理の途中段階の PV の情報の更新する処理段階のミスを発見した。このミスが分かったのは、ランダムなプロセスの故障を発生させたときであった。これは、プログラムの開発途中段階でも Neko のシミュレーション機能が使えたことがミスの発見につながったのである。これが、全てのプログラムが完成してシミュレーションを行っていたのでは、開発途中段階の複数のミスが起因して分からなかったかもしれない。特に大規模な分散システムのためのプログラム開発では、Neko のシミュレーション機能が必ず必要となるだろう。

第6章　評価

評価は、Neko のシミュレーション上で Chandra&Toueg の合意プログラムと Lazy consensus プログラムの比較をした。Lazy consensus は、Semi-passive replication を支えているコンポーネントである。また、Chandra&Toueg の合意アルゴリズムは、Active replication のコンポーネントになっている部分である。この２つのアルゴリズムを実装したプログラムを比較することで複製技術の基本的な部分の比較評価ができると考える。

評価で着目したポイントは以下のとおりである。

- プロセスの故障発生
- プロセスの数
- 合意実行回数

6.1　プロセスの故障がない場合の比較

最初にプロセスの故障が発生しない場合での Chandra&Toueg の合意アルゴリズムと Lazy consensus プログラムの比較をした[1]。このときの Heartbeat Failure detector のメッセージ送信時間幅は 40.12、制限時間は 60.12 とし、ネットワーク遅延時間はないものとする[2]。

表 6.1 と表 6.2 は、合意実行回数は 1 回とし、プロセスの数だけを変化させて２つの合意アルゴリズムを比較したものである。表 6.1 のプロセスの数が 5 と 10 のときでは、２つの合意アルゴリズムの実行時間に変化はない。しかし、プロセスの数が 20 以降は、変化がでてきた。この２つの表から考えられることは、２つの合意アルゴリズムの初期値に対する操作の違いである。Chandra&Toueg の合意アルゴリズムでは、全てのプロセスが合

プロセスの数	5	10	20	30	40	50
Consensus	14	15	58.12	82.12	103.12	177.12
Lazy consensus	14	15	25	35	45	55

表 6.1: 比較テスト 1

[1] このときの結果を示した表 6.1、表 6.2、表 6.3 の単位はミリセカンドとする。
[2] 単位はミリセカンドである。

プロセスの数	60	70	80	90	100
Consensus	209.12	241.12	510	573	637.84
Lazy consensus	179.12	199.12	222.12	243.12	263.12

表 6.2: 比較テスト 2

プロセス数：10						
合意実行回数	1	5	10	20	30	40
Consensus	15	2939	7476	15512	18385	26561
Lazy consensus	15	136	296	635	963	1313

表 6.3: 比較テスト 3

意実行に入るときに、必ず初期値を計算する。一方、Lazy consensus アルゴリズムでは、コーディネータだけが初期値を計算する。プロセスの計算処理の違いが合意実行の時間に影響していると思う。次にプロセスの数は変化させずに連続した合意実行の回数だけを変えて２つの合意プログラムを比較した（表 6.3)。

このときの比較では、２つの合意プログラムに大きな差が現れた。これだけの大きな差が現れた理由の１つは、Neko のシミュレーションで評価をしたことである。Neko のシミュレーションでは、プロセスのイベント処理は直列化されている。直列化とは、プロセスが１つのイベントしか処理できないことである。つまり、メッセージの受取りが同時にできないのである。もう１つの理由は、Heartbeat failure detector の時間設定（メッセージ送信時間幅と制限時間）である。合意実行回数によってハートビートメッセージが制限時間内にそれぞれのプロセスに届かないのである。これは、合意問題では、最適な Failure detector の時間設定が必要であること、また、２つの合意アルゴリズムのコーディネータリストの操作が大きく影響していると思う。特に故障に弱い分散 pl システム上でネットワーク遅延時間、メッセージの２重操作など Failure detector がプロセスが故障した可能性を探知する要因が多く存在する。複製技術を実装するときには、これらの要因に左右されない Failure detector を実装することが大切ではある。しかし、Failure detector の性質に左右されない合意アルゴリズムを複製技術のコンポーネントとして使う方が重要だと思う。これは、システム構築コストや複製技術の信頼性に大きく影響することだと考える。この点で、Lazy consensus アルゴリズムは、分散システム上に、複製技術を実装するときには適した合意アルゴリズムだと考える。

プロセス数：10			
合意実行回数	1	5	10
Consensus	15	3082	6915
Lazy consensus	14	73	154

表 6.4: ケース１の比較テスト

プロセス数：10			
合意実行回数	1	5	10
Consensus	15	3082	6971
Lazy consensus	14	83	171

表 6.5: ケース２の比較テスト

6.2　プロセスの故障がある場合の比較

プロセスの故障は、以下の３ケースで比較をした。ケース１と２では、毎回の最初の合意実行でのラウンド１でコーディネータとなるプロセスに故障を起こした[3]。

1. 値を提案する前の故障
2. 値を提案した後の故障
3. ランダムなポイントでランダムなプロセスの故障

表6.4、表6.5ともに合意実行回数が増加すると２つの合意プログラムの合意実行時間に大きく差が現れた。Chandra&Toueg の合意プログラムは、ケース１、２とも時間の差は、ほとんどない。しかし、Lazy consensus プログラムは、ケース１と比べて、ケース２の方がが少しではあるが時間の変化がある。これは、合意実行１回目のラウンド１で、コーディネータとなるプロセスが故障して、合意実行１回目において、１ラウンド多く費やしていることを表していると考えられる。

表6.6と表6.7は、合意実行回数10回で２つの合意プログラムを比較した。ケース３では、時間に大きなばらつきがあった。これは、ランダムなプロセスの故障が大きく影響し

プロセス数：5	
Consensus	376.2678
Lazy consensus	237.055

表 6.6: ケース３での最高タイムの比較

[3]表 6.4、表 6.5、表 6.6、表 6.7 の単位は全てミリセカンドである。

プロセス数：5	
Consensus	1309.865
Lazy consensus	1127.654

表 6.7: ケース３での最低タイムの比較

ていると考える。また、ケース３が、ネットワーク環境での評価環境に近いものと思う。ケース３はシミュレーション環境ではあるが、Lazy consesnsus プログラムの方が、ネットワーク環境での評価でも Chandra&Toueg の合意プログラムより良い合意実行時間が得られることが期待できる。

第7章　まとめ

この章では、Semi-passive replication のコンポーネントである Lazy consensus と本研究の特色である Neko についてまとめた。

7.1　Lazy consensus

本研究では、Semi-passive replication のコンポーネントである Lazy consensus を Neko に実装した。Lazy consensus アルゴリズムを実装したプログラムと Chandra&Toueg の合意アルゴリズムを実装したプログラムを比較することで、Lazy consensus には次の利点が分かった。

- プロセスの故障に影響されにくい
- 連続した合意処理に適している

合意処理がプロセスの故障に影響されにくいということは、複製サービスを要求するクライアントに故障の影響を与えないことにつながる。これは、分散システムの上に複製技術を実装するときにもっとも重要なポイントである。また、連続した合意処理に適しているので、複製サーバーに負荷を与えずにスムーズに複製サービス要求処理を行うことが期待できると考える。実際の複製サービスでは、複数のクライアントからの複製サービスが複製サーバーに要求される。Lazy consensus では、複数のクライアントに対しても待ち時間を与えずに要求処理を行うことができると考える。上記の Lazy consensus の利点は、分散システム上で、複製技術を実装するときに適したコンポーネントであると思う。

7.2　Neko

本研究では、分散アルゴリズム実装および開発環境に Neko を使った。Neko を使うことで、一人でも多くの時間と労力を必要とする分散アルゴリズムの実装を行うことができた。また、Neko が持つシミュレーション機能が筆者のプログラミング能力の助けとなった。そして、分散アルゴリズムのための豊富なコンポーネントが筆者が一人で分散アルゴリズムを実装することを可能にした思う。また、これら豊富なコンポーネントを読むことで、本の上でのアルゴリズムとプログラムの違いに気付いて、筆者が実装したプログラムに、その気付いたことを活かせたと思う。

謝辞

最後になりましたが、私がこの論文を書くにあたり力を貸してくださった皆様にこの場を借りて感謝を述べたいと思います。私がこの研究をするにあたり、私に複製技術について 1 年以上に渡り、ご指導くださった Défago 先生ありがとうございました。そして、文章の書き方の基礎を教えて下さいました青木先生ありがとうございました。これからも青木先生がご指導下さったことを忘れずに常に読み手が理解しやすい、しっかりとした文章を書くことに努めます。それから、この論文締め切りのぎりぎりまで付き合って下さったポスドクの林原さん、ありがとうございました。1 人で不安な気持ちでこの論文を書いている時のアドバイスが私を救ってくれました。そして、最後になりましたが、特に何の技術も知識もない私を、こんなすばらしい研究室に在籍することを認めて下さった片山先生ありがとうごいざいました。片山先生の研究室で研究させていただいたことを自分の誇りとして、これからも精進していくことに努めます。また、JAIST の全ての先生方、学位申請や研究発表に関する事務処理をして下さった全ての事務職員の皆様、本当にありがとうございました。

参考文献

[1] R.Guerraoui & A.Schiper, "Software-based replication for falut tolerance," *IEEE Computer*, 30(4):pp.68–74, April 1997.

[2] P.Urbán, X.Defágo & A.Schiper, "Neko : A Single Environment to Simulate and Prototype Distributed Algorithms," JOURNAL OF INFORMATION SCUENCE AND ENGINEERING 18, p981–997, 2002.

[3] X.Défago, A.Schiper, & N.Sergent, "Semi-passive replication," In textslProc. 17th IEEE Intel. Symp. textslReliable Distributed Systems(WORDŚ99), p43–50, West Lafayette, IN USA, Oct.1998.

[4] Micheal D.Schroder, "A State-of-the Art Distributed System:Computing with BOB," In S.Mullender(ed.),*Distributed Systems,*Addison Wesley, Second edition, 1993, pp.1-16.

[5] T.D.Chandra and S.Toueg, "Unreliable failure detectors for reliable distributed systems," textslJ.ACM43(2),pp225-267, 1996.

[6] M.J.Fischer, N.A.Lynch, and M.S.Paterson, "Impossibility of distributed consensus with one faulty process," *J.ACM*32(2),pp374-382, Apr.1985.

[7] Naohiro Hayashibara, X.Défago, Rami Yared, and Takuya Katayama, "The ϕ Accrual Failure Detector," *The* 23^{rd} *IEEE International Symposium on Reliable Distributed Systems(SRDŚ04).*

[8] N.Budhiraja, K.Marxullo, F.B.Schneider, and S.Toueg, "The primary-backup approach," In S.Mullender, editor, /textslDistributed Systems, ACM Press Books, chapter8, pp199-216, Addison-Wesley, second edition, 1993.

[9] F.B.Schneider "Replication Management using the state-Machine Approach," In S.Mullender, editor, /textslDistributed Systems, ACM Press Books, chapter7, pp169-195, Addison-Wesley, second edition, 1993.