情報処理学会東北支部研究会（２００３年第１回）
Ｂ－１－１

ソフトウェアの高信頼化によるパフォーマンスの低下を抑える方式の検討と実装

Design and Implementation of a Mechanism to Avoid the Performance Degradation in Highly Reliable Software

石井保（日大院・工・情報）　　杉山安洋（日大工・情報）

1. はじめに

オンラインシステムやサーバシステムなどの重要なシステムは、障害やシステムのメンテナンスによる一時的なシステムの停止ですら人間生活に与える影響は大きいものとなる。そこで、そのようなシステムのサーバ側の信頼性を向上させるために、サーバ側で使用されている分散オブジェクトシステムが停止することなく動作するシステムが必要になる。代表的な分散オブジェクトの技術として RMI や CORBA などがあり、本研究では RMI を使用する。RMI はクライアント側にスタブと呼ばれる代理オブジェクトを使用し、代理オブジェクトがネットワーク処理を行うので、クライアントはネットワーク処理についてそれほど意識せずに動作することができる。

我々の研究室では、RMI の概念を拡張し、RMI サーバをミラーリングすることにより実行中のソフトウェアを停止せずに運用するシステムとして、MMI（Mirrored Method Invocation）[1]を開発している。MMI のオブジェクトのミラーリングとは、一つの RMI サーバの複製を、複数のサーバマシン上で同時に実行しておき、それらを同じ状態に保っておくことにより、たとえ一台の RMI サーバで故障が起きても、その他 RMI のサーバが実行を続けられるようにしたメカニズムである。

本稿では、現在の MMI に存在する、パフォーマンスの問題を改善する事を目的とする。

2. MMI の概要

2.1 初期バージョンの MMI

MMI では Proxy というクライアントとサーバの仲介をするオブジェクトを使用する。図 1 のように Proxy はクライアントからのサーバに対する要求を最初に受け取り、すべてのサーバに対して要求を伝える。各サーバは渡された要求を処理し、処理結果を返す。その各サーバからの処理結果は Proxy が一旦全て受け取り、複数ある処理結果の中から一つの処理結果をクライアントへ返す。

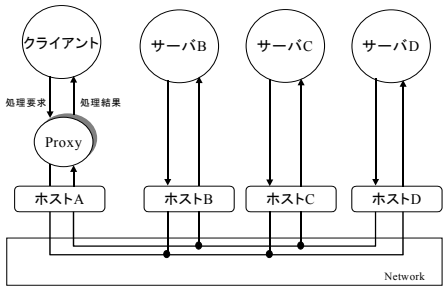

図 1　MMI の概要

例えば、クライアントからの要求をサーバ B、C、D に伝え、各サーバで処理していたとする。この時、サーバ B、C では処理結果が早い段階で返ってきているのにもかかわらず、サーバ D では何らかのトラブルや、コンピュータの処理能力のため、処理結果を返す時間が他のサーバより大幅に遅れてしまうという場合が考えられる。このような状況では、最も遅いサーバの処理結果が返ってくるまでクライアントに処理結果を返すのが遅れてしまう。その結果、全体としての処理結果が最も遅いサーバの処理速度に依存してしまう。サーバが停止してしまった場合には、そのサーバからの処理結果を待っていると、全体として止まった状態になってしまう。この問題を回避するため、初期バージョンの MMI ではプロキシ内で固定のタイムアウト時間を設定している。クライアントがサーバに要求を出した時刻を基準として、そのタイムアウト時間内に処理結果を返せなかったサーバは異常が発生しているとみなす。そして、その段階でサーバからの処理結果を待つのをやめて、次のクライアントの要求に移る（以後、タイムアウトをかけると呼ぶ）。ここで、タイムアウトをかけられたサーバは、何らかの問題が発生していると判断し、そのタイムアウト以降呼び出さない切り離し処理を行う。

2.2 MMI に存在する問題点

初期バージョンの MMI では、前述の通り固定のタイムアウト時間を使用している。この固定のタイムアウト時間を使うと、サーバの処理の大きさに関係なく遅いサーバや速いサーバに対して一つの同じタイムアウト時間を使用してしまうことになる。このため、元々タイムアウト時間を超えた時間を必要とする処理の場合、処理が終わる前にすべてのサーバに対してタイムアウトをかけてしまうことになる。元々のサーバの処理時間が、タイムアウト時間内にある場合でも、全体としての処理速度は、タイムアウト時間まで遅くなってしまうという問題がある。

また、切り離し処理に関して、タイムアウトをかけられたサーバは、何らかの問題が発生している可能性が高いという理由で、そのタイムアウト以後切り離しをしていた。しかしながら、遅くても正常に動作しているサーバも存在する。このため、タイムアウトをかけられたサーバに対して、単純に切り離し処理を行ったのでは問題がある。

3. タイムアウト処理

3.1 履歴を用いた処理時間の予測

履歴を用いた処理時間の予測方式は、各サーバを呼び出してから処理結果が返って来るまでの時間を記録しておき、その履歴から次にそのサーバがどれくらいの処理時間で処理結果を返す事ができるかを予測する方式である。この方式では、各サーバのメッソド毎に次の処理結果が返ってくるまでの時間（以下、予測時間と呼ぶ）を

設定する。これには、図２のように各サーバのメソッド毎の処理時間を、メソッドを呼び出す度に記録していき、その平均の処理時間を二割増しなどに増やした時間を予測時間として用いる。従って、各サーバのメソッド毎に予測時間を設定することができたるめ、各サーバのメソッド毎に可変で適切な予測時間を設定することができる。

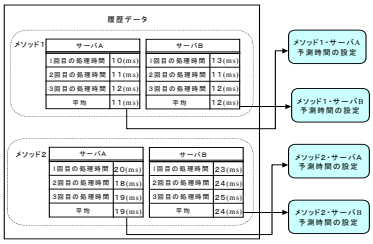

図２　履歴の適用方法

この方式は、履歴を用いていることで以後の適切だと思われる予測時間を求められ、履歴の処理時間が多くなればなるほど安定した予測時間を導き出せる。

3.2　同じ処理結果の個数による回答時間

全てのサーバからの処理結果を待っている場合、最も遅いサーバの平均処理時間に依存してしまうことになる。全てのサーバからの処理結果が返って来るまで待たなくても、いくつか処理結果が返ってきた時点でクライアントに処理結果を返し、次の処理を続ける方が効率的である。そこで、クライアントへ処理結果を返す時間（以下、回答時間と呼ぶ）を設定する。この回答時間を決定する際には、信頼性を低下させない程度に処理時間を向上させる回答時間を求める必要がある。

この回答時間に、同じ処理結果の個数による方式を提案する。これは、同じ処理結果がいくつか返ってきた時点で、タイムアウトをかけ、その処理結果をクライアントに返して次の処理を行うものである。例えば、図３のように最初に返ってきた処理結果と、その次に返ってきた処理結果の比較を行い、同じ処理結果であればクライアントに返す。そうでない場合は、同じ処理結果が二つ以上返ってくるまで待つ。同じ処理結果が二つ返ってくるまで待つ場合、通常は二つの処理結果だけ待てばよく、サーバの中に他のサーバより遅いサーバが存在する時にパフォーマンスは大幅に向上すると考えられる。また、信頼性に重点をおく場合は、同じ処理結果を待つ個数を増やすことで信頼性を損なわずにすむ。

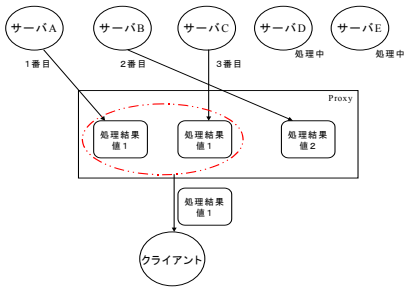

図３　処理結果の個数を用いたタイムアウトの例

例えば、同じ処理結果を二つ待つ場合、二つ目の処理結果が一つ目の処理結果の到着より大幅に遅く、二つ目の処理結果と最も遅い処理結果との差がほとんどない事も考えられる。その状況だと、パフォーマンスは最も遅いサーバを待っている場合と大差なくなってしまう。そこで、3.1 章の予測時間から次の処理結果が到着する時間が大幅に遅いことが予測できる場合は、そこでタイムアウトをかけることにより、パフォーマンスの低下を抑えることができる。

4.　遅いサーバの処理

遅いサーバは、タイムアウトの対象になることが多いが、処理が遅いだけで正常に処理を行っていることもある。そこで、タイムアウトにかかったサーバを単純に切り離すのではなく、そのサーバから処理結果返ってきた時に、その処理結果がクライアントに返した処理結果と違う場合に限り切り離すことにする。そして、遅いサーバに対しても、その後の要求を出せる機能を追加する。

4.1　キューの必要性

回答時間によるタイムアウトをかけられたサーバは、その時の処理が終わっていないため、処理結果をクライアントに返す事ができない。その後、そのサーバの処理が終了した時の処理結果は、クライアントに返した処理結果と同じ場合と違う場合がある。その各々の場合で処理を分けることにする。

クライアントに返した処理結果と同じ場合は、そのサーバは正常に動作していると判断する。遅くても正常な処理結果を返せるサーバは切り離さずに残しておいたほうが、処理できるサーバが減らないため、信頼性の低下を防ぐことにつながる。このため、クライアントに返した処理結果と同じ処理結果を返したサーバは、切り離し処理を行わないことにする。切り離しをしないのなら、そのタイムアウト以降に発生するクライアントからの処理要求を、発生した順に実行できる機能が必要になる。そこで、プロキシの中でクライアントからの処理を溜めておくキューを用いることで、その機能を実現する。

また、クライアントに返した処理結果と違う場合は、他のサーバと異なる状態になっている。それ以後、そのサーバに対して処理要求を出したとしても、状態が異なっているので、他のサーバと違う処理結果を返す危険性を含むことになる。このため、クライアントに返した処理結果と異なる処理結果を返したサーバには、それ以降呼び出さない切り離し処理を行うことにする。

4.2　キューによる要求制御

MMI でのキューは図 4 のように、各サーバに一つのキューを割り当てることにした。例えば、サーバ A、B、C が起動しているとすると、サーバ A 用のキュー、サーバ B 用のキュー、サーバ C 用のキューを作る。クライアントの要求が発生すると、その要求を対応するキューにエンキューする。要求をサーバへ伝えた時は、その要求はまだキューの先頭に残っている。サーバの処理が終わった時、その要求はキューから削除さる。要求が削除された後、後ろに次の要求が入っている場合は、その後ろの要求が先頭の要求となる。

図 4 を用いてキューの動作を説明する。ここでは、速いサーバ A、B と、遅いサーバ C で処理を行っている時に、次の状態になったと仮定する。まず、サーバ A、B、

C に同時に最初の要求を伝える。サーバ A、B は処理が終わり、回答時間によるタイムアウトでサーバ C はタイムアウトがかけられる。その後、クライアントから二つ目の要求が発生し、サーバ A、B は二つ目の処理を行う。サーバ C は、まだ一つ目の処理を実行中である。その後、サーバ A、B で二つ目の処理が終わり、三つ目の要求が発生する。サーバ C は依然一つ目の処理を実行している。以下でその時の動作を説明する。

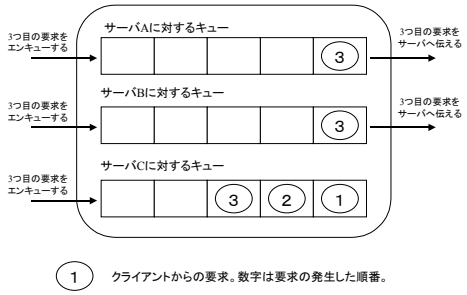

図４　キューの概要

クライアントが、三つ目の要求を出した時、サーバ A、B では一つ目と二つ目の要求の処理は終わっているため、三つ目の要求がキューの先頭に入り、三つ目の要求をサーバに伝えることができる。しかし、サーバ C は処理が遅いためタイムアウトをかけられ、まだ一つ目の処理が終わっていない状況にある。ここで、三つ目の処理要求がサーバ C にエンキューされた場合、要求は、発生した順番通りキューの一番後ろに入れられる。

サーバ A、B に対しては各サーバに処理要求を発生させ、サーバ A、B からの処理結果を待つ。その後、処理結果が返ってきたら、3.2 節で述べた処理結果の個数による方式を使い比較を行う。クライアントに処理結果を返せる状態になり、クライアントに処理結果を返したら、キューからその時の要求を削除する。

タイムアウトをかけられたサーバ C に対しては、サーバ A、B とは別に独立して処理が継続されることになる。一つ目の処理が終わったら、その要求をキューから削除し、キューに溜まっている次の処理を行う。これをキューの要求がなくなるまで繰り返す。キューから要求がなくなった時は、他のサーバに追いついたということである。

このキューの機能を実現することで、順番通り要求をサーバに伝えることができるようになった。

5.　実現方式

5.1 履歴管理オブジェクトの実装

履歴を用いるために、Proxy 内に履歴を管理する Career オブジェクトを作成し、各サーバのメソッド毎の履歴をそのオブジェクトが一箇所で管理するようにした。Career オブジェクトでは、行がメソッド、列がサーバとなる平均値の履歴テーブルを持たせた。Career オブジェクトには以下の三つの機能がある。

(ア) 処理時間を履歴テーブルに登録。
(イ) 各サーバのメソッド毎に予測時間を計算。
(ウ) 予測時間の取得要求に対して、その時間を渡す。

他のオブジェクトから Career オブジェクトに対して（ア）と（ウ）を呼び出せるようにしている。(イ) は (ア) により処理時間が追加された後に、予測時間を計算する。

5.2　キューの実装

4.2 節で述べた機能を持つキューは、Queue クラスとして実装した。このクラスのオブジェクトは、クライアントからの要求を最初に受け取る位置にある。このクラスには二つの機能を持たせた。一つは、キューの機能である。もう一つは、各サーバからの処理結果を比較し、同じ処理結果の判定をしてクライアントへ処理結果を返す機能である。

5.3　全体としての動作

Career オブジェクトと、Queue オブジェクトのプロキシ内での動作の概要を表したものが図 5 になる。

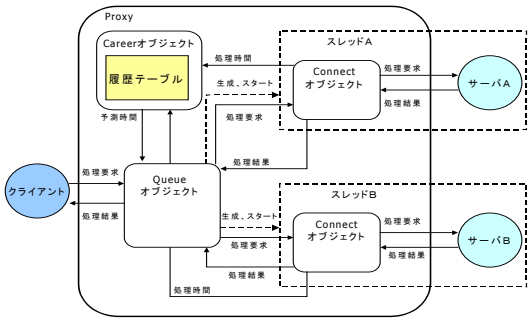

図５　全体の動作の概要

全体の動作として、図 5 のようにクライアントが処理要求を発生すると、Queue オブジェクトで、各サーバに対する要求をサーバ毎のキューにエンキューする。そのエンキューをした要求が、キューの先頭にある場合、Queue オブジェクトがそのサーバの対応メソッドに要求を渡すためのスレッドを作成する。すると、その各スレッドがキューから要求を取り出し、処理時間の測定を開始し、サーバに処理要求を出す。サーバが処理を終えたら、測定した処理時間を Career オブジェクトへ登録し、その処理結果を Queue オブジェクトに渡す。

Queue オブジェクトでは、返ってきた処理結果を溜めておき、サーバから処理結果が返ってくるたびに比較を行う。ここで、同じ処理結果が二つになったら、処理結果を返していないサーバに対してタイムアウトをかけ、その処理結果をクライアントへ返す。この時に、クライアントへ返した処理結果と違う処理結果を返したサーバに対しては、以後要求を出さない切り離し処理を行う。そして、クライアントから次の要求を受けられる状態にしておく。

また、タイムアウトをかけられたサーバのスレッドは、対応するサーバの処理が終わるのを待ち、処理結果が返ってきたら Queue オブジェクトに渡す。Queue オブジェクトでは、クライアントに返した処理結果を記憶しているので、その記憶してある処理結果とタイムアウトにかかったサーバからの処理結果を比較し、異なる処理結果であった場合は切り離し処理を行う。同じ処理結果だった場合は、キューからそのときの要求を削除して、キューに溜まっている次の要求をサーバに伝え、キューに

要求がなくなるまで続ける。

6. 関連製品

システムの信頼性を向上するためのシステムは既にいくつか存在している。その中でよく用いられるのは、クラスタリングシステムである。これは、複数のコンピュータを相互に接続することにより、ひとつのクラスタを構築し、ユーザーや他のコンピュータに対してそのクラスタを一台のコンピュータであるかのように振舞わせる方式である[2]。これにより、複数のコンピュータを一台のコンピュータを扱うように管理することができる。

今回､商品化されたクラスタリングシステムを調査し、詳細を検討する。

6.1 Sun Cluster

Sun Cluster[3][4]は、複数の Solaris システムをクラスタ化して集中管理し、データのバックアップや復元、ソフトウェアのアップグレード、新しいハードウェアの追加などのシステム管理作業を行うシステムである。このシステムでは、個々の Solaris システムがクラスタノードとなり、それぞれのノードやハードウェア、ソフトウェアが正常かどうかを常に監視している。アプリケーションを実行中のノードに障害が発生した場合、そのノードからは、サービスの配信やデータにアクセスできなくなる。この場合、障害の発生していない別のノードにデータやリソースなどの所有権を引き継ぎ、アプリケーションを再起動することになる。この処理をロールバックと言う。

ロールバックには以下の問題がある。あるトランザクションの途中で障害が発生し、アプリケーションが再起動されると、その時、途中まで実行していたトランザクションを無効化してしまう。つまり、トランザクションを最初からやり直さなければならないということである。クラスタリングシステムのロールバックは、アプリケーション単位でのロールバックとなり、無効化される処理が大きなものとなっている。このため、サービスが一時的に停止してしまうことになり、ユーザーはサーバに障害が発生したことを意識させられてしまう。

6.2 HP 9000 HyperPlex

HP 9000 HyperPlex[5]は、多数の HP9000 サーバをクラスタノードとして相互に接続している。このシステムは、ハードウェアとソフトウェアによる障害から基幹となるアプリケーションを保護することを目指すシステムである。このシステムの中に、イベントを監視するサービス（EMS）がある。CPU やシステムメモリ、アプリケーションなどが正常の状態であるか連続的に監視する。障害が発生すると、イベントを生成し、EMS に通知する。このイベントを連続的に監視することで、障害の検出を迅速に行っている。EMS は、通知されたイベントをもとに、障害の発生しそうなプロセッサを予測する Dynamic Processor Resilience 機能を備えている。これにより、障害が発生する前に、問題のプロセッサを切り離すことにより、致命的なエラーの発生を防ぐことができる。しかし、予測が失敗し、システムに致命的なエラーが発生した場合には、ダウンタイムの発生は避けられない。

7. まとめと今後の課題

今回実装したシステムにより､正常に動作しているが、遅いサーバに対しても、全てのクライアントからの要求を発生した順番通り伝えることが可能となった。このため、信頼性の低下を防ぐことができたと言える。また、回答時間に処理結果の個数による方式を適用したことにより、パフォーマンスの低下を回避することができた。

課題として、今回のシステムでは予測時間に平均処理時間を用いているのだが、処理時間のばらつきが大きい場合、適切な予測時間が求められない可能性がある。この問題の解決のため、処理時間の分布を用いる方式を提案する。この方式は、メッソド毎に各サーバの処理時間がどのように分布しているかを調べ、処理時間が密集している部分があるなら、その時間を予測時間として使用する方式である。分布の形態が正規分布の形態に近い場合には、平均時間を使用する。分布のばらつき状況を判定するには、標準偏差を使用する。予測時間では、標準偏差の値により、ばらつきが大きいと判断した場合、平均時間を使用してしまうと、その時間より遅い部分の処理時間に対して考慮していないことになる。このため、設定する予測時間が速くなりすぎる危険がある。そこで、予測時間を大きくし、遅い部分も含む時間に設定することでその危険性を回避する。そして、この方式について検討と実装を行い、実行時間を測定して評価する必要がある。

参考文献

[1]杉山安洋:分散オブジェクトの高信頼化へのアプローチ、ソフトウェア工学の基礎Ⅸ、2002

[2]嚴建泰:ソフトウェアの信頼性を向上させる方式の検討、平成 12 年度日本大学工学部学術研究報告会講演要旨集、2001

[3]Sun Cluster3 アーキテクチャ:Sun Cluster、http://jp.sun.com/products/software/serverperf/clusters/index.html

[4]Antonio Road:Sun Cluster Architecture:A White Paper 、1999 IEEE International Conference on Cluster Computing、1999

[5]Hewlett Packard Company:HP Hyperplex Clustering Technology、1999 IEEE International Conference on Cluster Computing、1999