5K−4

オープンシステムを用いたフォールトトレラントプラットフォーム PREGMA について*

三島健[1] 増田悦夫[1†] 藤田昭人[2] 土方陽一[2] 小野直俊[2‡]
日本電信電話株式会社 NTT ネットワークサービスシステム研究所[1]　NTT アドバンステクノロジ株式会社[2]
email : mishima.takeshi@lab.ntt.co.jp

1 はじめに

近年，インターネットを利用したサービスの発展に伴い，サービスを提供するサーバにはますます高い可用性が要求されている．そこで，高可用性を実現するために様々なフォールトトレラントシステム (FT)[2][3] やクラスタ [4] が提案されてきた．しかし，FT は専用のハードウェアや OS を必要とするため非常にコストが高いという欠点がある．一方，クラスタは PC サーバなどオープンシステムを用いるためコストは安いが，障害時のサービス中断時間が数分以上かかること，障害時に提供されていたサービスやそのサービスで作成された情報が失われることなどの問題点も多い．

そこで，筆者らは低コストと高可用性を両立するためにゆるやかな同期方式を提案しておりさらに本方式を用いたプラットフォームとして PREGMA を提案している [1]．本稿では，試作結果をもとに PREGMA の有効性を示す．

2 PREGMA の構成と特徴

2.1 構成

PREGMA は，実際にクライアントの要求を処理しサービスを提供するサーバと，サーバ間で同期を取る機能や障害検出，障害対処を行う機能を持つ Coordinator (CDR)，ハードウェアの障害を検出するための System Monitor (SM)，サーバの障害を検出するための Package Monitor (PM) から構成される (図 1)．

CDR は，サーバとクライアント間のメッセージを送受信する際にサーバの同期を取り，冗長化されたサーバ間の状態の一致を保証するゆるやかな同期方式を実現する [1]．また，CDR はサーバから受信したメッセージ同士の比較やメッセージの正当性をチェックすることによって障害の有無を判定する．

SM は主にハードウェアの障害を検出するために利用する．SM は定期的に *alive* メッセージを CDR に送信し，CDR はこれを受け取ることによって PC ハードウェアは正常であると判定する．もし，ある一定時間以内に *alive* メッセージを受け取ることができなかった場合には，その SM が存在する PC ハードウェアには何らかの障害があると判定する．一方，PM はアプリケーション (APL) の障害を検出するために利用する．PM は定期的にサーバの生存を確認し，死滅している場合にはサーバに障害があったことを CDR に報告する．

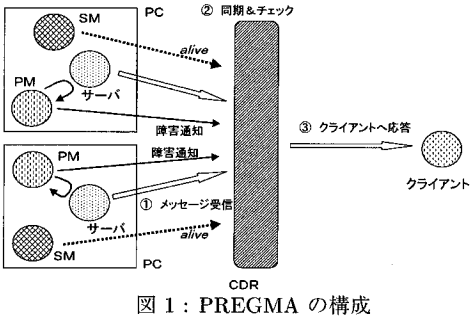

図 1：PREGMA の構成

CDR は，ゆるやかな同期時に障害を見つけた場合や，SM または PM によって障害を見つけた場合には，障害となったサーバ，PC を切り離し，クライアントには障害を隠蔽する．

2.2 特徴

従来の FT は特殊なハードウェアを用いて冗長化されたハードウェア間の同期を取っていたが，PREGMA ではプロセス単位で同期を取るので特殊なハードウェアは必要ない．従って，汎用 PC ばかりでなく，汎用の OS までも無改造で用いることができるため，低コスト化が実現できる．また，冗長化されたサーバ同士は同一のサービスを実施しているため，障害によりサーバが死滅したとしても一つのプロセスさえ生存していればクライアントへのサービスを継続することができる．この際，生存サーバは，情報を全く失わずに瞬時にサービスを継続できるので，従来のクラスタよりも非常に高い可用性を実現できる．さらに，障害に備えてサービスや情報の冗長化，障害の検出，障害時のサービス継続処理などは全て CDR が行なうため，APL には PREGMA の機能は透過である．従って，クラスタのようにスクリプトの開発や API を使った APL 開発は必要ない．また，FT のように特殊な OS で動作するように APL を開発，改造する必要もない．

---

*Fault Tolerant Platform PREGMA based on Open Systems
†Takeshi MISHIMA, Etsuo MASUDA
‡Akito FUJITA, Youichi HIJIKATA, Naotoshi ONO
[1]NTT Network Service Systems Laboratories, NTT Corporation
[2]NTT AdvancedTechnology Corporation

3 試作機の評価と考察

いかなる時にもクライアントにサービスを提供することは非常に重要なことであり，例え障害発生時においてもサービス中断時間は 0 秒に近ければ近いほど望ましい．文献 [1] における評価では，障害時の PREGMA のサービス停止時間は約 500 ms で実現できることを示した．本評価では，さらに 0 秒に近づけるための最適な *alive* メッセージ周期 t_{alive} とタイムアウト値 $t_{timeout}$ を評価し，その場合における PREGMA のサービス中断時間を評価する．

試作機で使用した PC のスペックは，CPU : Pentium III 850MHz, Memory : 512MB, OS : Linux kernel 2.2.16 であり，APL は Apache を使用した．また，CDR の *alive* メッセージチェック周期 t_{check} は，`usleep()` システムコールを使って約 20 ms としている．

3.1 最小 t_{alive} の評価

まず，$t_{timeout} = 5000$ ms (PREGMA にとっては十分大きな値) とした場合，t_{alive} はどれくらい小さくしても正常に動作できるかを評価した．その結果，負荷関係なく，$t_{alive} < 20ms$ で PREGMA のスループットが低下することを確認した．これは，$t_{alive} < t_{check}$ とすると t_{check} が約 20 ms であることから CDR がチェックすべき *alive* メッセージが溢れてしまうことが原因であると思われる．従って，最小 t_{alive} は 20ms である．

3.2 最小 $t_{timeout}$ の評価

次に，t_{alive} はある一定の値を設定した場合，$t_{timeout}$ はどれくらい小さくしても正常に動作できるかを評価した．図 2 は，横軸 t_{alive}，縦軸は障害が無いにもかかわらず障害があると誤判定してしまう直前の $t_{timeout}$ をプロットした．なお，低負荷の場合も高負荷の場合も $t_{timeout}$ の値は変わらないため，グラフは高負荷の場合を示す．図 2 より

$$t_{timeout} = t_{alive} + \alpha \tag{1}$$

という関係式が成り立つことが分かる (この場合，$\alpha = 30ms$ である)．α は，タイムアウト時間以内に CDR が *alive* メッセージを必ずチェックするために最低限必要な時間であるから，

$$\alpha = t_{check} + \delta \tag{2}$$

である (この場合，$\delta = 10$ である)．よって，(1)(2) 式より

$$t_{timeout} = t_{alive} + t_{check} + \delta \tag{3}$$

となる．従って，(3) 式と 3.1 より最小 $t_{timeout}$ は，50 ms である．

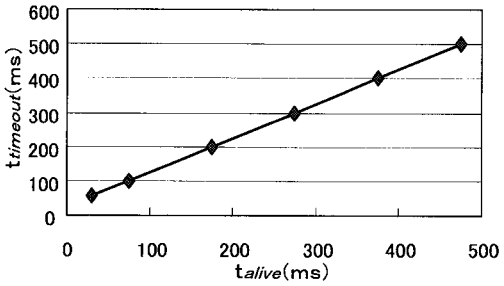

図 2 : t_{alive} と $t_{timeout}$ の関係

3.3 サービス中断時間の最小値

サービス中断時間の最小値 t_{min} は，障害検出時間 t_1 と障害切り離し時間 t_2 の和だから

$$t_{min} = t_1 + t_2 \tag{4}$$

となる．文献 [1] より $t_2 = 12ms$ だから $t_1 = t_{timeout}$ とすると $t_{min} = 62ms$ となり，障害時における PREGMA のサービス中断時間は非常に小さいことが分かった．

4 おわりに

本稿では，PREGMA アーキテクチャの概要を述べるとともに，核となるフォールトトレランス機能について試作結果に基づきその有効性を示した．実際のシステム化に当たっては CDR の高信頼化も必要であり，この点については別途報告予定である．

参考文献

[1] 三島，増田，藤田，土方，小野，“オープンシステムを用いたフォールトトレラントプラットフォームの提案，” 電子情報通信学会 2001 秋ソサイエティ大会，2001

[2] 金川，山口，福丸，宮尾，“フォールトトレラントサーバ FT6100 の高信頼化技術，” 信学技報，FTS93-56, pp.49-54, 1993

[3] 岡本，阿部，伊藤，曽我，“冗長構成のマルチプロセッサにおける回復方法について，” 信学技報，FTS94-32, pp.37-44, 1994

[4] 富川，田崎，藤原，“GRANPOWER 7000 クラスタシステムの設計思想と新技術，” 情処全大春，1D-1, 1998