Net Disk アーキテクチャにおける Global Address Manager の設計

趙　延紅† 掛下　哲郎††

† 佐賀大学工学系研究科　〒 840–8502 佐賀市本庄町 1 番地
†† 佐賀大学理工学部　〒 840–8502 佐賀市本庄町 1 番地
E-mail: †zhaoyh@cs.is.saga-u.ac.jp, ††kake@is.saga-u.ac.jp

あらまし　我々は高性能データベース・サーバを構成するために，Net Disk アーキテクチャを提案している．このシステムはディスクアレイの高負荷時にも効率的な動的負荷分散が行える．また，システムの耐故障性に優れている．本稿では，Net Disk アーキテクチャの構成要素である Global Address Manager (GAM) の設計を行った．そのために，設計の前段階として必要なメッセージ形式を定義する．また，メッセージを Bus 幅に合わせて分割送信する．次に，GAM の仕様として入出力，データ形式および GAM の内部データ構造を定義する．これに基づいて，GAM のアルゴリズムを作成する．また，テストケースを設計する．

キーワード　並列ディスク，動的負荷分散，アーキテクチャ設計，テストケース設計

Design of the Global Address Manager in the Net Disk Architecture

Yanhong ZHAO† and Tetsuro KAKESHITA††

† Department of Information Science, Saga University　1, Honjo, Saga, 840–8502 Japan
†† Faculty of Science and Engineering, Saga University　1, Honjo, Saga, 840–8502 Japan
E-mail: †zhaoyh@cs.is.saga-u.ac.jp, ††kake@is.saga-u.ac.jp

Abstract　We have proposed the Net Disk architecture in order to construct high-performance database servers. This system can efficiently perform dynamic load balancing even though the disk arrays are under heavy load conditions. The architecture also achieves high fault tolerance. In this paper, we design the Global Address Manager (GAM) in the Net Disk architecture. At first we define the message structures. Messages are splitted to transmit through the fixed size bus. Next we define GAM specification together with the internal data structure. Based on these, the algorithms of GAM are developed and the test cases are designed.

Key words　Parallel disks, Dynamic load balancing, Architecture design, Test case design

1. ま え が き

CPU は年 60%の速度向上があり，主記憶は年 100%の集積度向上がある．これに対して，ディスクの速度向上は年 7%以下で，CPU，主記憶とディスクの速度差が年々拡大している．従って，コンピュータシステムの主要なボトルネックはディスクであるといえる．しかし，ディスクは低コスト，高信頼性という点から，データベースシステムには不可欠なものである．そのため，データベース・サーバなどの大規模記憶装置の高速化，高信頼化に対する要求は年々高まってきている．

この問題を解決するために注目されているのが並列ディスクである．並列ディスクは複数のディスクを同時に動作させることで通常のディスクの性能をディスク台数に比例して向上させようとするものである．

しかし，並列ディスクは，ディスク台数の増加に伴い信頼性が低下する，データアクセス頻度の時間変化によりディスク間で負荷が偏るといった問題がある．

並列ディスクの問題点に対して，RAID [1], [2] (Redundant Array of Independent Disks) が考案されている．RAID は，並列ディスクのアクセス速度向上と，信頼性の向上を同時に図るために考察された手法である．RAID は安価なディスクを複数用いた並列ディスクシステムによって，高価なディスク装置に匹敵する記憶容量とアクセス速度を得ようとする．RAID では，信頼性の低下を防ぐためにデータを重複格納したりするなど，ディスク内のデータに冗長性を持たせている．RAID には，それぞれ冗長性の持たせ方が異なる RAID1 から RAID5 まで 5 種類の RAID が存在する．RAID ディスクは，いろいろな長所を持つために並列ディスクを構成する時にしばしば使われている．なかでも RAID1，RAID3，RAID5 はよく使われている．

RAID はディスク間での動的負荷分散が保証されない．そこ

で，本研究室ではこれらの研究に基づいて先に動的データ再配置 [3] を提案した．動的データ再配置は，並列ディスクの負荷分散を効率良く行うために考案された手法である．動的データ再配置では，負荷最大のディスクに格納されているデータがアクセスされると，ディスクはアクセス要求に対する応答を返すとともに，そのデータが負荷最小のディスクに動的に移動する．

本研究室では動的データ再配置を利用してディスクアレイ間の負荷分散を実現する Net Disk アーキテクチャ [4] を提案している．Net Disk アーキテクチャは，複数のディスクアレイ，アドレス変換を行うモジュール，動的負荷分散を制御するモジュール，そしてそれらを結合するクロスバネットワークによって構成されている (図 1 および表 1)．Net Disk アーキテクチャでは，ディスクアレイ内の動的再配置とディスクアレイ間の動的再配置を組み合わせてシステムの負荷バランスを保つ．

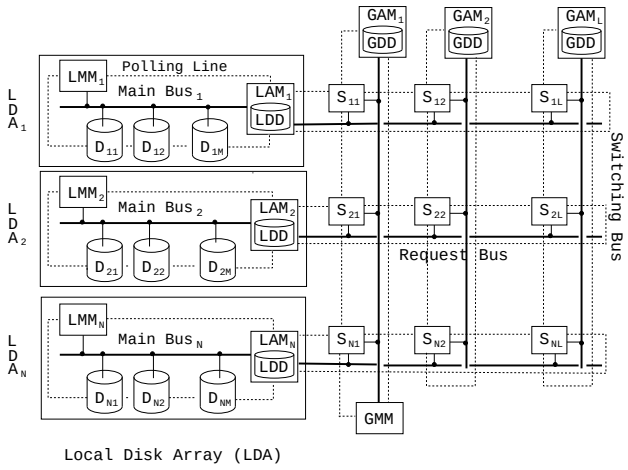


図 1　Net Disk アーキテクチャ

表 1　Net Disk アーキテクチャを構成するモジュール

略称	英語
GAM	Global Address Manager
GDD	Global Data Dictionary
GMM	Global Migration Manager
S	Switch
LAM	Local Address Manager
LDD	Local Data Dictionary
LMM	Local Migration Manager
D	Disk
LDA	Local Disk Array

本稿では，Net Disk アーキテクチャの実用化に向け，Net Disk アーキテクチャの構成要素である GAM の設計を行った．我々はこれまでにスイッチの論理回路の設計を行った [5]，また，LMM, GMM の設計を行った [6], [7]．スイッチは 2 つのバス間でメッセージをやり取りする論理回路である．これに対して，LMM, GMM, GAM が行う処理はスイッチよりもはるかに複雑である．そのため，論理回路で構成するのは合理的でない．そこで，LMM, GMM, GAM はマイクロプログラミングを用いて実現する予定である．本稿ではそのために必要となる，GAM のデータ構造とアルゴリズムを設計する．また，GAM の設計の不備やあいまいさを発見するために，原因-結果グラフ [8], [9] を使ってテストケースを設計する．

本稿は以下のように構成されている．2 節では Net Disk アーキテクチャの構成および動作について紹介する。3 節では User-GAM 間のメッセージとクロスバネットワーク上を流れるメッセージを定義する．また，バス幅に合わせたメッセージの分割を行う．4 節では GAM の入出力線および内部データを定義する．5 節では GAM のアルゴリズムを作成する．6 節では有効なテストケース設計および無効なテストケース設計を行う．

2. Net Disk アーキテクチャの構成と動作

本節では，Net Disk アーキテクチャの構成および動作について述べる．

2.1 Net Disk アーキテクチャの構成

Net Disk アーキテクチャでは，外部からのアクセス要求はいずれかの GAM が受け付ける．データはディスク D_{ij} が分散して格納する．以下に Net Disk アーキテクチャを構成するモジュールについて説明する．

GAM　GDD を用いて，アクセス要求の論理アドレスを LDA 物理アドレスと LDA 内論理アドレスの組に変換する．そして，LDA に対し要求メッセージを送る．また，GMM から再配置開始命令を受け，LDA 間の動的再配置を実行する．

GMM　各 LDA の負荷数を収集する．その情報を元にして，GAM に対し LDA 間の再配置開始/停止命令を送る．

LDA　LAM，LMM，D によって構成される．それぞれのモジュールは Main Bus に接続されている．

LAM　送られてきた要求メッセージの LDA 内論理アドレスを，LDA 内物理アドレスに変換して当該ディスクにアクセス要求を送る．このモジュールでアドレス変換することで LDA 外部にデータ格納先の変更を意識させないようにしている．また，LMM から再配置開始命令を受け，LDA 内での動的再配置を行う．

LMM　Main Bus に流れるメッセージを監視し，各ディスクの負荷を求めている．この情報を元にして，LDA 内の再配置開始/停止命令を送る．

クロスバネットワーク　スイッチにより Switching Bus と Request Bus 間の接続を制御することで，各モジュール間の通信を行う．

Net Disk アーキテクチャを用いると，LDA 内では局所的な動的再配置を行い，LDA 間では大域的な動的再配置を行うことで，巨大なディスクアレイに対する動的負荷分散を実現する．Switching Bus や Request Bus は Main Bus と同一のデータ転送速度を持てばボトルネックにはならない．また，負荷の重いディスクまた LDA へのデータアクセス時に，そのデータの再配置を同時に行うので，システムの高負荷時にも効率的な動的負荷分散が行える点で優れている．Net Disk アーキテクチャは，バスやディスク等を多重化することでハードウェア面での耐故障性を持ち，高い信頼性を実現する．

2.2 Net Disk アーキテクチャの動作

Net Disk アーキテクチャにおける負荷分散は 2 つのステップによって達成される．これらは LDA 内の局所的な動的再配置と LDA 間の大域的な動的再配置である．GMM は大域的な動的再配置を制御する．動的再配置は，通常のデータアクセス処理と同時に実行する点が特徴である．そこで，本節では，通常の Read(Write) 処理の流れを説明した後で，大域的な動的再配置の動作を説明する．

通常時のデータアクセス処理

(1) 利用者からの UserRead (UserWrite) 要求は，GAM_i が受信し，論理アドレスを LDA 内の論理アドレスとデータが格納されている LDA の物理アドレスに変換する．

(2) GAM_i は $Switch_{ji}$ を経由して，Read(Write) メッセージを LAM_j に送る，LAM_j は受け取ったメッセージの論理アドレスを LDA 内物理アドレスに変換し，Read (Write) メッセージを当該ディスク D に送る．

(3) LAM_j がディスク D からメッセージを受け取ると，$Switch_{ji}$ を経由して，GAM_i に送り返す．

(4) GAM_i は回答メッセージを利用者に送る．

大域的な動的再配置の制御

各 LAM は LDA の負荷数を計算し，一定時間毎に GMM へ送る．GMM は負荷最大 (負荷最小) の LDA 番号 LDAMax (LDAMin) を決める．LDAMax の負荷数 / LDAMin の負荷数 $> \delta$ $(1 \leqq \delta)$ が成立すると，GMM からクロスバネットワークに再配置開始命令をブロードキャストする．これにより，すべての GAM が再配置モードになる．再配置モードになった GAM は大域的な動的再配置を実行する．LDAMax の負荷数 / LDAMin の負荷数 $\leqq \lambda$ $(1 \leqq \lambda < \delta)$ が成立すると，GMM から GAM に再配置停止命令をブロードキャストする．これにより各 GAM は通常モードに戻る．大域的な動的再配置時の Read (Write) 処理を図 2 に示す．

再配置モード時の Read 処理

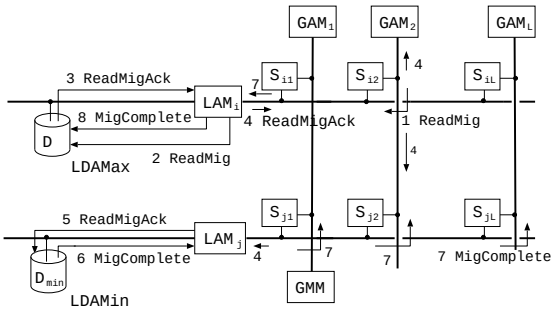
GAM_i から LDAMax に対する Read 要求を受けると，GAM_i は LDAMax に ReadMig メッセージを送る．LDAMax が ReadMigAck メッセージをクロスバネットワークにブロードキャストすると，そのメッセージは GAM_i と LDAMin が受信する．LDAMin はデータを保存した後，クロスバネットワークを経由して，MigComplete メッセージを LDAMax と GAM_i に送る．LDAMax はこれに応えて古いデータを消去する．GAM_i は MigComplete メッセージを受信した後，GDD を用いて，LDA の物理アドレスを変換する．

再配置モード時の Write 処理

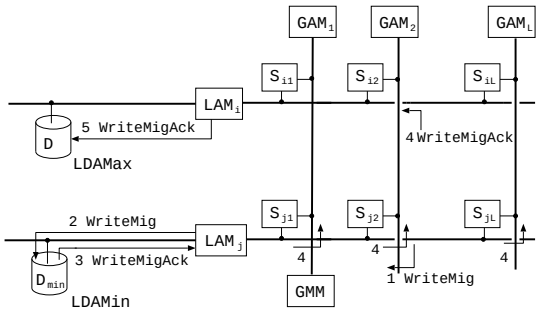
GAM_i から LDAMax に対する Write 要求を受けると，GAM_i は LDAMin に WriteMig メッセージを送る．LDAMin は送信されたデータを保存して，WriteMigAck メッセージをクロスバネットワークに送り返す．そのメッセージは LDAMax と GAM_i が受信する．LDAMax は古いデータを消去する．GAM_i は GDD を用いて，LDA の物理アドレスを変換する．

動的再配置は以下の三つの特徴を持つ．

- データが負荷最大の LDA から負荷最小の LDA に移動するため LDA 間の動的負荷分散を実現する．
- データアクセスとデータの再配置を同時に行うためクロスバネットワークや LDAMax のオーバーヘッドが少ない．
- しばしばアクセスされるデータが再配置されやすいので，最小限のデータ移動で負荷分散を実現できる．



(a)Read Operation with Global Migration



(b)Write Operation with Global Migration

図 2　大域的な動的再配置時の Read (Write) 処理

3. メッセージのデータ構造

User-GAM 間を流れるメッセージのデータ構造，クロスバネットワークを流れるメッセージのデータ構造は，GAM の仕様を決定する上で必要となる．ここで，2 節で述べた Net Disk アーキテクチャの基本動作より，メッセージのデータ構造を定義する．

3.1 User-GAM 間のメッセージ

Net Disk アーキテクチャにおいて，User-GAM 間を流れるメッセージを表 2 に示す．メッセージの種類を 3 ビットで定義する．User-GAM 間でやり取りされるメッセージにおいて，メッセージ内容をメッセージ先頭ワードとメッセージ終端ワードで囲み，分割してメッセージの送受信を行う．ここで，メッセージ先頭ワードを定義する．32bits 幅のバスに合わせて，メッセージ先頭ワードは表 3 に示すように三つのワードに分割する．データ部分はメッセージ先頭ワード送信後に送信される．メッセージ終端ワードのメッセージ内容は，全て 1 とする．

3.2 クロスバネットワーク上のメッセージ

Net Disk アーキテクチャにおいて，クロスバネットワーク上を流れるメッセージを表 4 に示す．メッセージの種類を 5 ビッ

表 2　User-GAM 間のメッセージ

メッセージ名	略称
ユーザからの読み出し要求メッセージ	UserRead
ユーザからの書き込み要求メッセージ	UserWrite
ユーザへ読み出し要求結果メッセージ	UserReadAck
ユーザへ書き込み要求結果メッセージ	UserWriteAck

表 3　User-GAM 間のメッセージの先頭ワード

ワード	ビット	名称	概要
1	3	MessageType	メッセージの種類
	4〜7	GAMAddress	GAM 番号
2	8〜39	Address	データの論理アドレス
3	40〜71	MessageNumber	メッセージの番号

トで定義する．User-GAM 間のメッセージと同様に，クロスバネットワークを流れるメッセージを分割して送受信を行う．メッセージ先頭ワードは表 5 に示すように三つのワードに分割する．

表 4　クロスバネットワーク上のメッセージ

メッセージ名	略称
再配置開始命令	MigStart
再配置停止命令	MigStop
再配置完了命令	MigComplete
通常時読み出し要求結果メッセージ	ReadAck
再配置時読み出し要求結果メッセージ	ReadMigAck
通常時書き込み要求結果メッセージ	WriteAck
再配置時書き込み要求結果メッセージ	WriteMigAck
通常時読み出し要求メッセージ	Read
再配置時読み出し要求メッセージ	ReadMig
通常時書き込み要求メッセージ	Write
再配置時書き込み要求メッセージ	WriteMig

表 5　クロスバネットワーク上メッセージの先頭ワード

ワード	ビット	名称	概要
1	1〜5	MessageType	メッセージの種類
	6〜9	LDAAddress	LDA 番号 (通常時) 負荷最大 LDA 番号 (再配置時)
	10〜13	LDAMinAddress	負荷最小 LDA 番号
	14〜17	GAMAddress	GAM 番号
	18〜21	LDALAddress	LDA 内論理アドレス
2	22〜53	Address	データの論理アドレス
3	54〜85	MessageNumber	メッセージの番号

4.　GAM の仕様

本節では 2 節で述べた Net Disk アーキテクチャの構造と動作，3 節で述べた User-GAM 間を流れるメッセージおよびクロスバネットワークを流れるメッセージのデータ構造をもとにして，Net Disk アーキテクチャにおける GAM の仕様を決定する．仕様では，入出力線とデータ形式および内部データを明確にする．

GAM の入出力を図 3 に示す．MessageFlag はメッセージ先頭ワード/終端ワードとデータ部分を識別するために用いる．

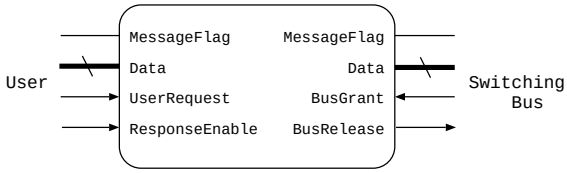


図 3　GAM の入出力

User 側の信号線のうち UserRequest, ResponseEnable は，GAM がユーザから受け取る要求メッセージおよびユーザへと返す要求結果メッセージを送受信するために用いる．

Switching Bus に接続されている信号線のうち BusGrant や BusRelease はポーリングによるバス調停を実現するために用いる．Switching Bus には GAM 以外にも GMM 等のモジュールが接続されているので，これらの間でバス調停を行う必要がある．各モジュールは BusGrant を通じてバス利用権を受け取った時に，メッセージを Switching Bus にブロードキャストできる．ブロードキャストが済むと，BusRelease を通じて隣のモジュールにバス利用権を渡す．

GAM の各信号線の詳細を表 6 に示す．GAM の内部データの詳細を表 7 に示す．データ形式は正規表現で記述している．| は選択構造を，∗ は繰り返し構造をそれぞれ表す．

5.　GAM の設計

本節では GAM の仕様に基づいて設計を行う．最初に GAM の全体構造を示す．GAM の機能は複雑なので，マイクロプログラミングによって実現する予定である．そこで (1) GAM のシステムパラメータの初期設定，(2) ユーザおよび Switching Bus から受け取ったメッセージの受信処理，(3) ユーザおよび Switching Bus への送信処理についてアルゴリズムを構築する．

5.1　GAM の全体構造

GAM の全体構造を図 4 に示す．

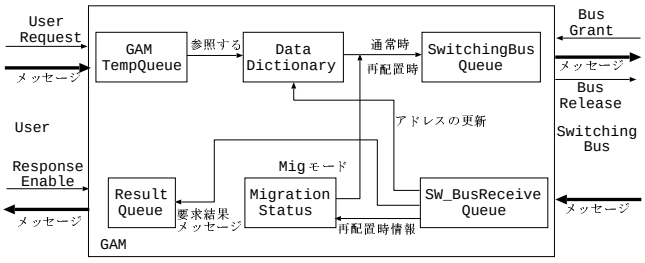


図 4　GAM の全体構造

ユーザ側の UserRequest が 1 になった場合は，要求メッセージを GAMTempQueue に一時保存する．DataDictionary を参照して，データの論理アドレスを LDA 内論理アドレスと LDA 番号に変換する．通常時また再配置時のメッセージの種類に変換する．その後メッセージを SwitchingBusQueue に保存する．

BusGrant が入力されると，SwitchingBusQueue に保存されているすべてのメッセージを Switching Bus にブロードキャストする．

表 6　GAM の入出力線とデータ形式

信号線	Input/Output	データ形式	概要
MessageFlag	I/O	[0 | 1]	メッセージ開始/終了フラグ/データ
Data	I/O		メッセージ内容は先頭ワード，各種データが記述される．終端ワード時には，全ての Bits が 1 となる．
UserRequest	I	[True | False]	ユーザが GAM にメッセージを送る命令.
ResponseEnable	I	[True | False]	ユーザが GAM からメッセージを受け取る命令.
BusGrant	I	[True | False]	バス利用権の取得
BusRelease	O	[True | False]	バス利用権の解放

表 7　GAM の内部データ

名称	データ形式	概要
GAMTempQueue	(メッセージ)*	ユーザから受信したメッセージを一時保存する.
DataDictionary	(データの論理アドレス + LDA 番号 + LDA 内論理アドレス)*	各データの論理アドレスおよび LDA 番号，LDA 内論理アドレスの対応表を管理する.
SW_BusReceiveQueue	(メッセージ)*	Switching Bus から受信したメッセージを保存する.
MigrationStatus	(MigFlag + LDAMax + LDAMin)	再配置情報を格納する.
SwitchingBusQueue	(メッセージ)*	Switching Bus にブロードキャストするメッセージを保存する
ResultQueue	(メッセージ)*	ユーザへ送る要求結果メッセージを一時保存する.

Switching Bus 側からのメッセージを SW_BusReceiveQueue に保存する．メッセージが MigStart ならば，再配置情報を書き込む．メッセージが MigStop ならば，通常モードに戻る．メッセージが MigComplete ならば，GDD を更新する．メッセージが要求結果メッセージならば，ユーザ側のメッセージの種類に変換する．その後メッセージを ResultQueue に保存する．

ResponseEnable が 1 になった場合は，ResultQueue に保存した要求結果メッセージを利用者に送る．

5.2　システムパラメータの初期設定

(1) GAMTempQueue, SW_BusReceiveQueue, SwitchingBusQueue, ResultQueue を空にする.

(2) MigrationStatus に MigFlag を 0 にする.

(3) DataDictionary をハードディスクから GAM の主記憶にロードする.

5.3　メッセージの受信処理

GAM はユーザおよび Switching Bus からのメッセージを受信して処理する．メッセージの処理手順はそれぞれ以下のようになる．

ユーザ側

ユーザが送信したメッセージ (UserRead, UserWrite) は，GAMTempQueue に一時保存され，必要な変換を施した後で SwitchingBusQueue に転送される．GAMTempQueue や SwitchingBusQueue へのデータの書き込みや読み出しはワード単位で行われる．

(1) UserRequest が 1 になると，ユーザが送信したメッセージの各ワードを順次読み込んで GAMTempQueue に書き込む.

(2) GAMTempQueue が保持する当該メッセージの種類，GAM 番号，データの論理アドレス，メッセージの番号 (表 3) を読み込む.

(3) DataDictionary を使ってデータの論理アドレスを LDA 番号と LDA 内論理アドレスの組に変換する.

(4) 通常モードの場合，または再配置モードかつ上記の LDA 番号が負荷最大の LDA 番号 (LDAMax) と一致しない場合には，クロスバネットワーク上の先頭ワード (表 5) を生成して，SwitchingBusQueue に書き込む (通常時のデータアクセス)．ここで，メッセージの種類 (表 4) は Read または Write とする．なお，LDAAddress には LDA 番号を，LDAMinAddress には空をそれぞれセットする.

(5) そうでない場合にはクロスバネットワーク上の先頭ワード (表 5) を生成して，SwitchingBusQueue に書き込む (動的データ再配置)．ここで，メッセージの種類 (表 4) は ReadMig または WriteMig とする．なお，LDAAddress には LDA 番号 (注：MigrationStatus が保持する LDAMax と一致する) を，LDAMinAddress には負荷最小の LDA 番号 (MigrationStatus が保持する) をそれぞれセットする.

(6) 先頭ワード以降の各ワードを SwitchingBusQueue に順次書き込む.

(7) GAMTempQueue から当該メッセージを削除する.

Switching Bus 側

Switching Bus 上を流れるメッセージ (MigStart, MigStop, MigComplete, ReadAck, WriteAck, ReadMigAck, WriteMigAck) は SW_BusReceiveQueue に一時保存され，MigrationStatus, DataDictionary, ResultQueue に必要なメッセージを書き込む．SW_BusReceiveQueue や ResultQueue へのデータの書き込むや読み出しはワード単位で行われる．

(1) Switching Bus 上を流れるメッセージの各ワードを順次読み込んで SW_BusReceiveQueue に書き込む.

(2) SW_BusReceiveQueue が保持する当該メッセージの種類 (表 4) を読み込む.

(3) メッセージの種類が MigStart ならば，以下の処理を行う.

(3-1) MigFlag を 1 にする.
(3-2) LDAMax, LDAMin を更新する.

(4) メッセージの種類が MigStop ならば，MigFlag を 0 にする.

(5) メッセージの種類が MigComplete ならば，DataDictionary に対して主記憶と二次記憶に保持している LDA 番号を更新する.

(6) メッセージの種類が ReadAck, WriteAck, ReadMigAck, WriteMigAck のいずれかならば，以下の処理を行う.

(6-1) 当該メッセージの GAM 番号，データの論理アドレス，メッセージの番号を読み込む.
(6-2) メッセージの種類を User-GAM 間の形式 (表 3) に変換する.
(6-3) User-GAM 間のメッセージを生成して ResultQueue に書き込む.

(7) SW_BusReceiveQueue から当該メッセージを削除する.

5.4 メッセージの送信処理

GAM はユーザおよび Switching Bus にのメッセージを送信して処理する．メッセージの処理手順はそれぞれ以下のようになる.

ユーザ側

ResultQueue に保存したメッセージは UserReadAck, UserWriteAck である．ResponseEnable が入力されると，ユーザへのデータの読み出しはワード単位で行われる.

(1) ResponseEnable が 1 になった場合，ResultQueue に保存されているすべてのメッセージをユーザに送信する.

Switching Bus 側

SwitchingBusQueue に保存したメッセージは Read, Write, ReadMig, WrieMig である．BusGrant が入力されると，Switching Bus へのデータの読み出しはワード単位で行われる.

(1) SwitchingBusQueue のすべてのメッセージを Switching Bus にブロードキャストする.

(2) BusRelease を 1 にする.

6. テストケース設計

GAM の設計の不備やあいまいさを見つけ出すためにテストケースを設計する．そのために有効なテストケースは原因-結果グラフを使って作成する．原因-結果グラフを用いた有効なテストケースの作成は，(1) 原因および結果の列挙，(2) 原因および結果の間の関連の識別，(3) 原因の可能な組み合わせが尽くされていることの確認，(4) テストケースの抽出の 4 ステップによって行う．最後に無効なテストケース設計を行う.

原因と結果の列挙

GAM の振る舞いに影響を与える原因としては，GAM に対する入力メッセージおよび GAM の内部状態が挙げられる．これらを表 8 に示す．入力メッセージはユーザからのもの (表 8 の 1〜4) と Switching Bus からのもの (表 8 の 6〜13) に分類できる．表 8 の 5 は GAM の内部状態を表す.

表 8 原因-結果グラフの原因 (入力と状態)

番号	概要
1	ResponseEnable を入力する
2	UserRequest を入力する
3	UserRead を入力する
4	UserWrite を入力する
5	GAM が再配置モードかつ LDAMax と要求対象 LDA が一致する.
6	MigStart を入力する
7	MigStop を入力する
8	MigComplete を入力する
9	ReadAck を入力する
10	ReadMigAck を入力する
11	WriteAck を入力する
12	WriteMigAck を入力する
13	BusGrant を入力する

一方，GAM の振る舞いを表す結果としては，GAM の内部状態の変化および GAM が外部に出力するメッセージが挙げられる．これらを表 9 に示す．表 9 の 81〜89 は GAM の内部状態の変化に対する．表 9 の 80 と 90 はそれぞれユーザおよび Switching Bus に対する出力である.

表 9 原因-結果グラフの結果 (出力と状態変化)

番号	概要
80	ResultQueue のメッセージをユーザに出力する
81	Read メッセージを SwitchingBusQueue に追加する.
82	ReadMig メッセージを SwitchingBusQueue に追加する.
83	Write メッセージを SwitchingBusQueue に追加する.
84	WriteMig メッセージを SwitchingBusQueue に追加する.
85	再配置モードに移行する
86	通常モードに移行する
87	GDD を更新する
88	UserReadAck を ResultQueue に追加する.
89	UserWriteAck を ResultQueue に追加する.
90	SwitchingBusQueue のメッセージを Switching Bus に出力する．その後，BusRelease を出力する.

原因および結果の間の関連の識別

表 8 および表 9 で列挙された原因および結果は独立に発生するものではない．例えば，表 8 の 1 と 2 が同時に発生することはない．表 8 の 3 と 4，6〜12 についても同様な排他制約が成立するので，図 5 ではこれを E (Exclusive) で表す．また，表 8 の 2 は 3 および 4 が成立するための前提である．図 5 では，このような制約を R (Require) で表す．原因と結果の間には因果関係がある．これを表すために図 5 では原因 (節点 1〜13)，中間節点 (節点 30〜40)，結果 (節点 80〜90) を枝で結び，必要に

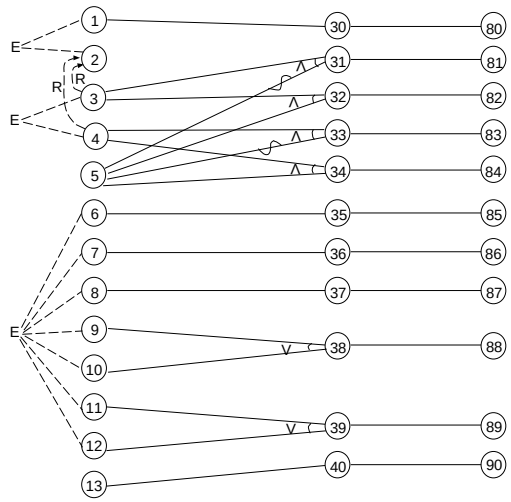


図 5　原因-結果グラフ

応じて AND, OR, NOT で結合している．

原因の可能な組み合わせが尽くされていることの確認

図 5 の原因-結果グラフで可能な原因の組み合わせが全て尽くされていることを確認するために，原因-結果グラフを以下に示す三つの連結成分に分割する．

1. 節点 1〜5，30〜34，80〜84
2. 節点 6〜12，35〜39，85〜89
3. 節点 13，40，90

これらの連結成分は互いに独立な原因を持つ．また，排他制約と前提制約を考慮すると，連結成分毎に可能な原因の組み合わせを尽くしていることが容易に分かる．

テストケースの抽出

テストケースの抽出は図 5 の連結成分毎に行う．原因-結果グラフから互いに独立な原因 (の組み合わせ) と結果の組を抽出すると，それをテストケースに変換できる．以下の 13 通りの有効なテストケースが考えられる．

(1) ResponseEnable を入力すると，ResultQueue のメッセージをユーザに送信する (原因：1，結果：80).

(2) UserRead を入力すると, Read を SwitchingBusQueue に追加する (原因：2，3，5，結果：81).

(3) UserRead を入力すると, ReadMig を SwitchingBusQueue に追加する (原因：2，3，5，結果：82).

(4) UserWrite を入力すると, Write を SwitchingBusQueue に追加する (原因：2，4，5，結果：83).

(5) UserWrite を入力すると, WriteMig を SwitchingBusQueue に追加する (原因：2，4，5，結果：84).

(6) MigStart を入力すると，再配置モードに移行する (原因：6，結果：85).

(7) MigStop を入力すると，通常モードに移行する (原因：7，結果：86).

(8) MigComplete を入力すると，GDD を更新する (原因：8，結果：87).

(9) ReadAck を入力すると，UserReadAck を ResultQueue に追加する (原因：9，結果：88).

(10) ReadMigAck を入力すると，UserReadAck を ResultQueue に追加する (原因：10，結果：88).

(11) WriteAck を入力すると，UserWriteAck を ResultQueue に追加する (原因：11，結果：89).

(12) WriteMigAck を入力すると，UserWriteAck を ResultQueue に追加する (原因：12，結果：89).

(13) BusGrant が入力すると，SwitchingBusQueue のメッセージを Switching Bus にブロードキャストする．その後 BusRelease を出力する (原因：13，結果：90).

無効テストケースの設計

以上で原因-結果グラフを使った有効なテストケースが設計された．この他に無効なテストケースを追加することで，異常処理や例外処理に適用できるようになるので，GAM の信頼性が向上する．

無効テストケースを設計するために，GAM に対する異常な入力および内部状態を系統的に列挙する．

GAM に対する入力メッセージは表 2〜表 5 で定義されている．それぞれの表に従って無効な入力を列挙すると以下のようになる．

(1) 表 2 および表 4 で定義されている以外の命令コードが入力された．

(2) 存在しないアドレスが指定された．該当するアドレスとしては，GAM 番号，データの論理アドレス，LDA 番号，負荷最大 LDA 番号，負荷最小 LDA 番号，LDA 内論理アドレスがある．

(3) 存在しないメッセージ番号が指定された．

(4) 順序が規定されている命令がその通りの順序で到着しない．

(5) メッセージの終端ワードが到着しない．

なお，信号線 MessageFlag, UserRequest, ResponseEnable, BusGrant は 1bit で定義されるので，エラーはないと仮定している．また，データ部分は任意のビット列なので，エラーはないと仮定している．

GAM の入力メッセージの異常に対応するテストケースとしては，上記の各場合に対応して以下が挙げられる．このうち (1)〜(4) の場合には GAM は出力および状態変化を何も行わない．

(1) 定義されている以外の命令コードが入力された場合．

(2) GAM 番号が定義した以外のアドレスの場合．

(3) UserReadAck のメッセージ番号が UserRead のメッセージの番号と一致しない場合．

(4) UserRead 命令と対応しない UserReadAck が入力された場合．

(5) 終端ワードがない UserWrite 命令を入力すると，GAMTempQueue が一杯になるまで到着したデータを順次保存する．GAMTempQueue が一杯になると，それ以降のデータは無視する．

GAM の内部状態は，図 4 および表 7 の各データ構造で定義されている．それらのデータ構造の異常としては，以下の場合が考えられる．

(1) データ構造がいっぱいである．

(2) データ内容が互いに矛盾している．

GAM の内部状態の異常に対応するテストケースとしては，上記の各場合に対応して以下が挙げられる．

(1) SwitchingBusQueue のデータがいっぱいの場合，新しい要求メッセージは無視される．

(2) 再配置モードでありかつ LDAMin で指定した LDA が存在しない場合に，LDAMax に対する UserRead 命令が到着すると，ReadMig ではなく，Read を SwitchingBusQueue に追加する．

同様な手法を用いると，他の異常テストケースが設計できる．

7. あ と が き

本論文では，Net Disk アーキテクチャの実用化に向け，GAM の設計を行った．まず User-GAM 間のメッセージとクロスバネットワーク上を流れるメッセージのデータ構造を定義した．次に，GAM の入出力線と内部データを定義した．これらより，GAM のアルゴリズムを構築した．また，有効なテストケースおよび無効なテストケースを設計した．

今後は，マイクロプログラムの内容を理解しやすいために高級言語を用いて記述プログラムを作る．これに基づいてアセンブラ言語と同様にマイクロ命令と 1 対 1 の関係にある記述プログラミングを行なう [10]．また Net Disk アーキテクチャにおける LAM の設計を行う．

文　　献

[1] P. A. Patterson, G. Gibson, R. H. Katz, “A case for redundant arrays of inexpensive disks (RAID)”, Proc. ACM SIGMOD, pp. 109-116, 1988.

[2] アンドリュー・S・タネンバウム, “構造化コンピュータ構成 第 4 版”, 株式会社ピアソン・エデュケーション, 2000.

[3] T.Kakeshita, S.Kubo, “A transaction processing architecture for effective load balancing utilizing high speed bus”, Proc.Int.Symp.on Cooperative Database Systems for Advanced Applications, pp. 27-30, 1996.

[4] T.Kakeshita and S.Zhang, “The Net Disk architecture for dynamic load balancing among disk arrays”, In Processings of the 7th International Conference on Parallel and Distributed Systems ICPADS 2000 Iwate, Japan 4-7 July, 2000.

[5] 趙 延紅, 掛下 哲郎, “Net Disk アーキテクチャにおけるスイッチの論理回路設計”, 電気関係学会九州支部第 56 回連合大会講演論文集 09−1P−01.

[6] 趙 延紅, 掛下 哲郎, “Net Disk アーキテクチャにおける Local Migration Manager の設計”, 火の国情報シンポジウム 2004 予稿集, B−4−4.

[7] 趙 延紅, 掛下 哲郎, “Net Disk アーキテクチャにおける Global Migration Manager の設計”, 電気関係学会九州支部第 57 回連合大会講演論文集 09−1A−03.

[8] G. J. Myers, “ソフトウェア・テストの技法”, 近代科学社, 2001.

[9] 河村 一樹, “ソフトウェア工学入門”, 近代科学社, 2003.

[10] J. P. Hayes. “Computer Architecture and Organization”, McGraw-Hill International Editions, 1988.