RAS 技術の現状*

北村拓郎** 富田克一***

1. はしがき

計算機の出現以来，現在にいたるまで計算機の信頼性，稼動性，あるいは保全性をいかにして高めるかは，常にそれぞれの時代の重要な問題であり，大きな努力が払われてきた．しかしこの問題の取り扱われ方は，その時代の部品の信頼性，計算機システムの規模と使用法，時代の技術水準によって変遷を経てきている．ここでこの問題に関する経過を大型計算機を中心に簡単にふりかえってみよう．

計算機の歴史において，第一世代と呼ばれる時代は，リレーまたは真空管を使用した計算機の時代であり，1950 年代がこれにあたる．これらの主要部品は，計算機のように多数の素子を使用する機械にとってはいささか信頼性に欠けるものであった．このために，数数の誤動作検出回路，誤動作防止回路を設けて，使用に耐える装置を実現しようとした．この時代の計算機の使用法は，いわゆる科学計算を主として，現在からみれば，単純なものであったが，システムの信頼度を満たすために，部品の信頼度の低さを方式的な数々の工夫で補っていたといえるであろう．

第二の世代は，トランジスタを使用した計算機の時代であり，1960 年代の前半がこれにあたる．トランジスタの出現によって計算機システムの信頼度は格段に改善され，以前に多く設けられていた誤動作検出回路，誤動作防止回路は簡素化され，信頼性の向上は，方式上の工夫よりも部品の信頼度の向上で得られるもので十分と思われた．この時代に計算機の使用法は，いわゆる事務処理分野にひろく拡大されたけれども，バッチ処理を主体とするものであった．この時代の部品の信頼度は，計算機の使用法の要求を上まわって十分であったといえよう．

第三の世代は，IC（集積回路）を使用した計算機の時代であり，1960 年代の後半がこれにあたる．IC の出現によって部品の信頼度は以前にも増して向上したけれども計算機システムの規模が大きくなり，計算機の使用法が複雑となり，信頼性向上に対する要求もきびしくなるとともに多様化した．この時代には，大型計算機の使用法として，いわゆる on-line 処理が盛んとなり，複雑なオペレーティング・システムのもとでのマルチプログラミング運転が普及した．このような状況にあっては計算機システムの機能停止となるシステム・ダウンは，もっとも避けなければならないこととなり，故障からの早急な回復が，以前よりも増して重要なこととなった．

現在は，IC の集積度が徐々に上がっていく中で，第三世代のあとを受けて，第 3.5 世代とも呼ばれている．この世代のもっとも大きな特徴の一つは，RAS（信頼性・可用性・保全性）の重視であり，これらに関する具体的な技術上の手段が豊富となってきていることである．

本稿では，現在における RAS の概念について説明し，現在の内外の進歩した大型計算機にみられる具体的な方式上の手段，特徴について解説をしようとするものである．

2. 大型計算機と RAS

2.1 RAS の概念

RAS とは，Reliability（信頼性），Availability（可用性），Serviceability（保全性）の略号である．古典的な概念からすれば，これらはほぼ次のような尺度で表わされるものに対応しよう．

Reliability → MTBF（平均故障間隔時間）

Availability → 稼動率＝有効稼動時間/有効稼動時間＋保守時間＋無効時間

Serviceability → MTTR（平均故障修理時間）

システムが単純である場合には，古典的な概念によるこれらの概念だけで十分であるようにみえた．しかしシステムが大きくなり複雑になるにつれて，これら

---

* RAS Technology, by Takuo Kitamura (Data Communication Development Laboratory, Nippon Electric Co. Ltd.) and Katsuichi Tomita (EDP System-Engineering Division, Nippon Electric Co. Ltd.)

** 日本電気株式会社データ通信開発本部
*** 日本電気株式会社コンピュータ方式技術本部

の尺度の単純な適用だけでは十分でなくなってきた．

まず信頼性について考えるならば，故障間隔の故障とは何かということである．一部の故障があっても，システムに致命的でないようにすることができる場合と，システム全体が停止するような故障とは同列には論じられない．また，可用性について考えるならば，システムの障害のために生じたジョブの再実行のための時間を無効時間とするならば，この時間は，機械の使用法との関係が密接となってくる．また，保全性について考えるならば，単に故障機械の修理時間が短いだけがよいのでなく，システムに与える影響が少ないのが望ましい．このためには，システムが稼動しながら off-line 的に故障部分が修理される場合には，多少時間が長くてもよい場合も生ずる．一方，システムの運転面からみると，いかに少ない運転要員で自動的にシステムを運転でき，少ない保守費用と保守要員でシステムの稼動を維持できることが，可用性・保全性を考えるうえで重要なこととなる．

以上を要約して，現在の RAS の目標は次のようになる．第一義的には，Reliability については，システムダウン間隔時間を長くすること，Availability については，システム稼動率（システム・アップタイム比）を大きくすること，Serviceability については，システム・ダウン時間を少なくすることである．しかしこれらは，互いに独立でなく関係し合っているものとして総合的にとらえること．これらのことを，できるだけ少ない費用で，効率よく成しとげることが要請される．

2.2 RAS 技術

現在の RAS の目標は，上述したとおりであるけれども，その目標を実現するための具体的な技術について，現在の大型計算機の場合を中心に考えてみよう．

図1 は，RAS に関する各種のアプローチと RAS の各概念との関係を示したものである．図1のマトリックスで丸印のある場合は，横軸のアプローチが，縦軸の概念の改善を目標としていることを表わしている．部品信頼度の向上はまず取り上げられるべきアプローチであるが，システムの RAS に関する要求は，これだけに頼ることを許さない．部品レベルによる冗長度設計は，基本回路（アンド・ゲートやオア・ゲートなど）自身を二重化，三重化することによって信頼度を上げる方法である．この方法は，軍用その他の特殊目的以外では，経済性の点から，現在では用いられていない．装置レベルによる冗長度設計は，大はシステムから，小は装置の中の機能モジュールまで，装置レベルでの多重化をはかるものである．この方法は，現在，ひろく普通に用いられているものであるが，適用分野によって変化に富むものである．故障の自動検出，訂正の萌芽は古くからあったけれども，近年新たな装いとともに盛んとなってきたものである．自動診断とその関連技術は，次のような理由によって今後ますます発展が期待される技術である．計算機システムの普及とともに熟練した保守者の不足をきたすこと．複雑で大きなシステムでは，人間による故障診断は困難となること．

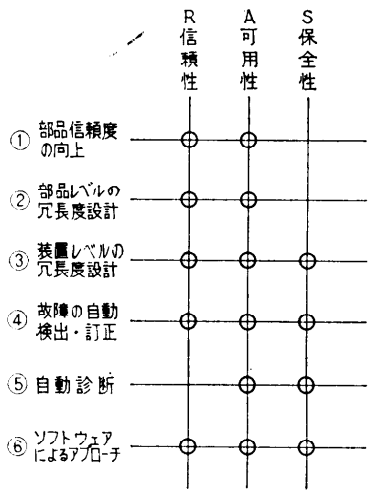

図 1 RAS 技術の構成

以上のような RAS 技術も計算機システムの利用技術の中に有効に組み込まれてはじめて効果を発揮する．この橋渡しの役割を果たすのが，オペレーティング・システムと呼ばれるソフト・ウェアシステムである．近年の大型計算機のオペレーティング・システムは，RAS 技術を駆使することを重要な目標の一つにしている．

以下の章では，部品信頼度の向上，部品レベル冗長度設計のアプローチを除いて，さらに詳しく上述の RAS 技術について解説しよう．

3. 冗長度による設計

ある装置やシステムの RAS が，単独の構成では，要求される RAS を満足しない場合に，正常な機能としては冗長性を持たせて，複数装置あるいは複数システムによるシステム構成をとり，RAS 要求を満たすようにシステム設計がなされることがある．ここでは

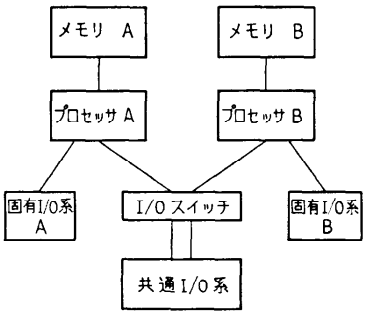

図 2 Duplex システム

大型計算機システムで通常，とられているいくつかのシステム構成について計算機本体を中心に述べよう．

3.1 Duplex システム

この方式は，もっとも簡便な方法でひろく用いられている．**図2** に Duplex システムの構成が示されている．計算機本体および記憶装置は，それぞれ独立に2組おかれ，1組が処理系として，固有 I/O 系と共通 I/O 系を使用してジョブを実行しているときは，他の1組は待機系となっている．ジョブ実行中に処理系に障害が発生すると，ジョブを中断して直ちにスイッチを切り替えることによって，共通I/O系を待機系に切り替え，ジョブの再開を行なう．この方式においては，処理系から待機系への切り替えは，手動で行なわれるのが普通である．また，待機系の利用率が低い．なお，プロセッサから共通I/O系に至る経路は二重化されているのが普通である．

3.2 Multi-Processor システム I（独立のモニタ・プログラムを有する場合）

この方式は，Duplex システムよりさらに進んで，ハードウェア的には，**図3** のように，2台のプロセッサで同一記憶装置を共有させたものである．通常には，2組のシステムがともに動作し，負荷を配分し合って処理を実行している．したがって，I/O 系に関するスイッチもダイナミックに切り替わる性質のものである．片方のプロセッサが故障の場合には，システムから自動的に切り離され，負荷は他のプロセッサに集中する．この方式においては，ハードウェア的には，互いの系は結合されているけれども，ソフトウェア的にはおのおのの系には独立のモニタ・プログラムが存在し，各ジョブ単位では，ほぼ独立の処理が行なわれている．このことによって，オペレーティング・システムとしてのソフトウェアが次に述べる方式に比べて簡単といえる．この方式は，Duplex システムに比べて，ハード・ウェア・リソースの使用法がより効率的であり，ジョブの中断，再開が自動的であることがすぐれている．しかし，そのための ソフトウェアを用意する必要がある．

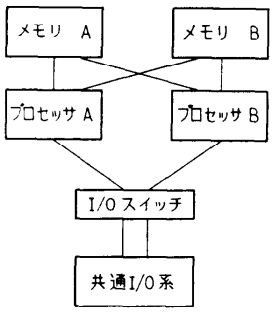

図 3 Multi-Processor システム

3.3 Multi-processor システム II（共通のモニタ・プログラムを有する場合）

この方式は，ハードウェア的な構成は図3の場合と同じであるけれどもソフトウェア的な使用法が異なる．システム全体には，システム全体の制御をつかさどるモニタ・プログラムは一つしか存在せず，計算機本体も，データ処理を実行するリソースの一つとみなされるにすぎない．ジョブ処理の中は，タスクと呼ばれるさらに細かいプログラムの単位に分かたれ，個々のタスクがどちらのプロセッサで実行されるかは，モニタの状態によって定められる．通常は，2台のプロセッサが動作して，システム全体では，2個のタスクが同時に処理されている．障害の場合には，障害を起こしたプロセッサは切り離され，同時に1個のタスクだけが処理される．この方式は，ハードウェア・リソースの効率的な利用という点で，また，3台以上のプロセッサのシステムへの拡張の点で，さらに進んだものであるが，一方，オペレーティング・システムが，さらに複雑なものとなること，場合によっては，複数タスクを同時に実行させるための余分な処理手続きが，オーバーヘッド・ロスとなって処理能率の低下となって現われることがあることなどが難点である．なおこの方式において，プロセッサを演算処理用，I/O 処理用，通信制御処理用，などのそれぞれの目的別に専用化する方法が用いられることもある．

4. 誤り検出と自動訂正方式

計算機の動作の誤りの検出は，チェック回路と呼ばれる部分で行なわれる．そして，ここで検出された誤

りの訂正 (error correction) が行なわれたり，命令レベルで再試行 (retry) が行なわれたりして，ハードウェア内部での処理が行なわれるのであるが，その各段階での方式を中心に説明する．

4.1 チェック回路

チェック回路の目的とするところは

① 原因の近くでの検出を行なう．

② 間欠的な障害をとらえる．

③ 計算結果の信憑性を高める．

などである．①によって，システムの一部の障害が他の部分にまで伝搬して，破局的システム・ダウンになるのを防ぐことができ，②によれば，間欠的障害のロギングを行なうことによって，これを統計的に解析し，障害が固定化する前に，予防することができる．

この目的に沿って，各種のチェック回路が設計され，装置内に組み込まれる．その量は，数パーセントから数十パーセントと設計によってまちまちであるが，チェック回路の増加分に対して，それでおおわれる障害検出可能部分の増加分は，一般にある点を越えれば急に減少する．したがって，すべての故障を100% 瞬間的にとらえるチェック回路を組み込むことはむずかしく，普通は行なわれない．

チェックの方式は各種あって，処理装置内のそれぞれの部分によって制御方式，回路構成などから，最適のものが選択される．ここでは，代表的な方式について紹介する．

（1） **parity-prediction** adder, counter, shifter など，データの操作を行なう回路に対して適用されるもので，データを操作する論理回路とは独立に，入力データのパリティビットを扱うパリティ論理回路を働かせて結果のパリティを作っておき，データ操作の論理回路より出力された結果のパリティとマッチングさせて，結果の信憑性を問うものである．

（2） **modulo-check** parity-prediction と同様の考えで，演算回路に用いられる方式で，入力データの modulo から，出力データの modulo を演算しておき，結果の modulo との一致をみる方式である．modulo としては，Adder 回路の構成上 2^n (n 任意) と素となる数が選ばれ，普通は3が選ばれる．

（3） ***k* out of *n* check** Sequence Control など制御回路に組み込まれて，論理の流れの正当性をチェックするものである．たとえば，論理回路を，同時動作しないブロックに分割して，それぞれの動作中にのみセットされるフリップフロップのビットを割り付けておく．これらのビットは常にたかだか1個しかセットされていないはずであるから，検出が可能となる．

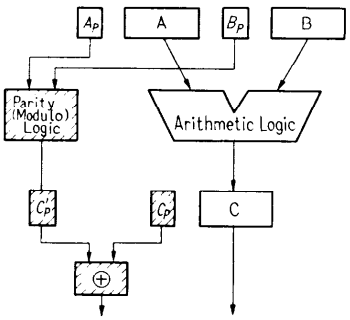

図 4 チェック付演算回路

その他，データの parity check，定義されていないコードをチェックする illegal code check，常に一定時間内にパルスが発生することを監視する time out check など，処理装置内の各論理回路によって，それぞれ固有のチェックが考慮されている．検出の結果は，制御レジスタに障害内容を記録するとともに，割込み動作でプログラムに障害が報告される．

4.2 再試行 (Retry)

障害には，固定故障 (solid failure) と間欠的故障 (intermittent failure) がある．間欠的故障は再試行によって回復することも考えられる．再試行が成功すれば，実行中のジョブを中断することなく続行させることができ，障害箇所はジョブの終了を待って適当な時点で修理すればよい．このようなことから再試行は非常に有効と考えられる．

命令の遂行には，一般に準備，演算，格納の段階がある．準備の段階で生じた障害は，もう一度最初から命令，オペランドの取り出しをやり直せばよいのであるから，容易に再試行可能である．演算の段階以降になると，元のオペランド（入力データ）が主記憶やレジスタにそのまま残っている保証がなくなるので，可能か否かは不確定である．つまり，命令の実行がどこまで進んだかによって再試行の可否が決まるが，この判定のために retry threshold と呼ばれるビットを付加し，障害発生時点でこのビットにより再試行を行なうか否かの判定を行なえるよう考慮する．さらに進んだ再試行の機能になると，オペランドを取り出した段階でその内容を演算回路内部で記憶しておき，どの時点で障害が起きても，再試行可能なように準備しておく方式もある．

4.3 自動訂正 (Error Correction)

障害を処理する手法には再試行とは別に，データ自動訂正がある．前者は上述のように，処理装置内の制御回路およびデータ処理において生じた障害に対する対策であり，同じ動作を繰り返すことによって回復をはかるものであるが，後者は，現在のところ，記憶装置内でのデータの誤り処理を対象としている．しかし時間関係とハードウェア量が許されるならば，データ転送一般に適用することのできる性質のものである．この方式によると，データの1ビットとの誤り箇所を検出し，自動的に訂正を行なうことができる．

データは冗長ビットが付されて error correction code として構成される．code 理論によれば，Humming code は 2^n ビットのデータに対して，$n+2$ ビットの付加ビットがあれば，一重の誤りを訂正し，二重の誤りを検出する能力があることがよく知られている．記憶装置の転送幅は，大型計算機の場合，多くは 64 ビットであるので，8ビットの付加ビットでよい．

5. 自動診断技術

障害が検出され自動訂正，再試行が行なわれるが不成功に終わったりもともとそういう手法の対象となっていない回路などについては，その処理を中断して，故障箇所を修理する必要がある．修理するためには，故障箇所がどこであるかを診断しなくてはならない．ここでは，その診断の一手法として，障害を含む装置を static にみて，これを自動診断する方式——一般に Fault Locating Test (FLT) と呼ばれる——について述べる．

FLT の原理は，対象とする論理回路の所定の入力に対する応答（出力）を，故障が全くない場合と，すべての故障を仮定した場合のおのおのについて求めておき故障したとき同じ入力に対する応答をみて（先に求めた出力データと比較する），故障箇所を指適するものである．これを処理装置の命令実行のための論理機能と独立して，所定の刺激パターン (stimulus pattern) を与え，その応答から組合せ回路的にチェックするものであり，入力（刺激パターン）は External Storage により自動的に行なわれる．以下，その方式について順を追って説明する．

（1） **scan-in, scan-out**　あらかじめ求められた刺激パターンを，External Storage より処理装置のフリップフロップにセット (Scan-in) したり，逆に，フリップフロップの状態を External storge へ導く (Scan-out) 機能がハードウェアに備わっていなければならない．これが non-functional に装置を診断するための不可欠な要素であって，この動作は，処理装置自身で行なわれたり，他の処理装置によって行なわれたりする．

（2） **stimulus pattern**　刺激パターンは直接，障害装置に対する入力データであり，この応答によって障害箇所を指摘するものであり，適切なパターンの選択によって，障害箇所を指摘する精度を上げることができる．あらかじめ被試験論理回路をいくつかの組合せ回路に分割し，そのおのおのが刺激パターンの対象となる．パターンは，設計自動化システムにより発生された論理マスターファイル（実装情報を含む）を処理して，プログラムにより自動的に発生される．

（3） **clock advance**　刺激パターンをフリップフロップに Scan-in したら，そのパターンのもとで動作させるために，命令動作のための，通常のクロックを1ステップ進める (Clock advance)．その結果（応答）を Scan-out して次の処理に移る．

（4） **結果の比較と故障箇所の指摘**　Scan-out されてきたデータと，あらかじめ求めておいた結果とのつき合わせを行ない．その結果によって障害箇所の摘出をする．

FLT では，普通一重のスタック故障を仮定して，パターンが発生される．一重のスタック故障とは，論理素子（アンドゲートやオアゲートなど）の入力または出力の1箇所が1または0に固定してしまう故障をいい，間欠的に生ずる不安定な故障や，多重故障は仮定されない．一重のスタック故障以外の故障は，Scan-in, out による Static な試験では検出される保証がない．

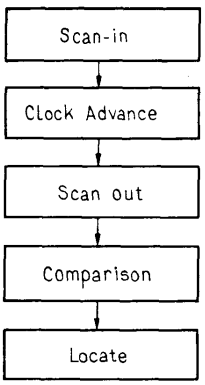

図 5　自動診断手順

6. ソフトウェア技術と RAS

これまでおもにハードウェアによる RAS 技術を述べてきたが，これと関連づけてソフトウェア側からの RAS を述べる．ここでは特に断わらない限り，ソフトウェアをオペレーティング・システムにおけるプログラムとして扱う．

6.1 Recovery と Reconfiguration

ハードウェアで検出された障害は，割込みという形でソフトウェアに報告され，同時に障害の内容が所定のレジスタに登録される．また，可能ならば，自動訂正や再試行が行なわれる．

検出された障害は，Recovery Management System (RMS) と呼ばれるプログラム・システテムで処理される．ここでは障害の解析，再試行，記録，診断，再構成などをつかさどる．ハードウェアによる自動訂正や，再試行が成功した場合，ここでは，その記録をエラー・ロギング・ファイルにしるすのみである．しかし，このファイルを定期的に統計，解析することによって，致命的障害にいたる前に予防できる．

障害割込が発生すると，RMS は retry threshold を参照して，ソフトウェア・レベルで再試行する（ハードウェアで自動的に再試行する方式もある．）．成功すれば記録のみにとどまり，不成功の場合は診断や再構成がなされる．診断は，システムをとめることなく，オンラインで診断プログラムを call して行なう．

その結果は解析されて保守者に報告される．診断の結果，修理をするために障害装置をオフラインにしたり，またそれに伴って代替の装置への切り替え，あるいは degrade など処理を続行できるようにシステムを再構成する．修理完了後も grade up のため再構成が行なわれ，高稼動率を維持して，障害による恒久的なサービスの低下を防ぐ．これらの障害処理手順は図 6 に示されている．

6.2 On-line Test

障害装置を診断する際，そのプログラムがシステムを占有する排他的なものであれば，プログラムはシステムが所定の処理を終わるか，またはサービスを中断しなければ診断も修理もできない．ただしプログラムが排他的でない場合はオペレーティング・システムのもとで診断を行なう方法が考えられる．オペレーティング・システムのもとで一般のジョブ・プログラムと並行して，診断プログラムにより行なう試験を on-linetest と呼ぶ．この運用については前述した．なお

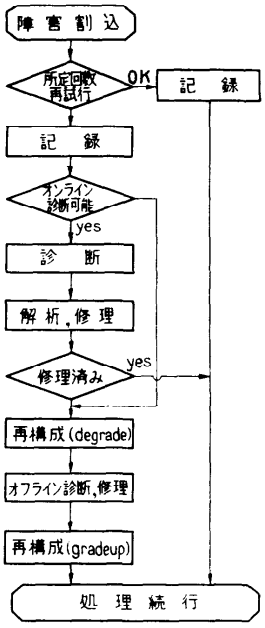

図 6 障害処理手順

on-line test プログラムは，修理後の確認テストにも使用できる．

6.3 Micro Diagnostics

Micro-programming 処理方式による金物部分については，Micro-instruction を用いた診断を行なうことができる．この場合診断に必要とされるマイクロ・プログラムは書き替え可能なマイクロ・オペレーション用の記憶装置に選択的に内部記憶装置から次々と読み込まれる．これによれば，通常の診断のためのハードウェアの試験やジョブ処理中における諸機能の試験が診断用の micro-instruction を programming しておくことによって，容易に行なえるようになる．普通のプログラムから，こうした診断用のマイクロ・プログラムへの切り替えが自動的に行なわれたり，一つのボタン操作のみでこうした試験が可能となるように工夫がはかられる．

6.4 DA と FLT

オペレーティング・システムとは別に，FLT システムをソフトウェアでサポートしているものに設計自動化システム (Design Automation System-DA) がある．DA は FLT で用いる刺激パターンや，locating のためのパターン，さらには，保守・診断のためのドキュメント作成情報などを発生させる源資となっている論理マスターファイルを提供する．これには，処理

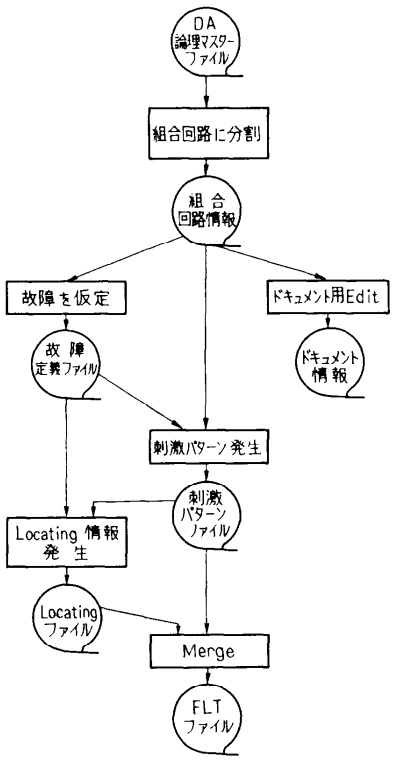

図 7　FLT パターン発生

装置におけるパッケージの実装位置やパッケージ間接続情報，また各パッケージ内での論理実装情報を含むものが用いられ，FLT パターン発生システムが固有の処理をほどこして，目的のものを発生させる．これらの処理の概略を示したものが図7である．

7. 今後の課題

近年，特に大型計算機の発達とともに，RAS 技術は重要な技術分野となっていることは前に述べた．しかし，体系的な技術としての歴史も浅く，今後の課題となることが多い．以下に，若干の今後の課題と思われるものをあげておこう．

（1） RAS の尺度と設計理論

RAS の尺度を明確にし，与えられた目標に対して，最適なシステムを設計できる設計理論を構築することが必要である．このときに，コストとパフォーマンスと RAS 目標とのトレード・オフについて，十分に検討されなければならない．

（2） フィールド・データの集積とフィード・バック

RAS 目標を実現するために設計されたシステムが実際に稼動している状況を，適切に測定し記録する方法を確立する必要がある．そうして集められた十分なフィールド・データに適切な分析を加え，将来の改善のために有効な指針を作成することが肝要である．

（3） 高信頼度システム

軍事利用などで多くの例がみられる高信頼度システム技術の一般商用機への適用の可能性について，常に十分に検討される必要があろう．

（4） 通信線の利用

いくつかの計算機が通信線によってネット・ワークを形成しているとき，通信線を利用してある計算機が他の計算機システムの稼動状況監視，保守診断を行なうことが考えられる．このときにある場所は，十分な資料とスキルを備えた Availability 制御センタのような役割を果たすことが考えられよう．

（5） 試験保守プログラムの自動発生

計算機の構造（論理回路やマイクロ命令）を与えたときに，試験精度，診断精度のよい試験または保守プログラムを自動発生させること．

（6） 間欠障害対策

間欠的に発生する障害の故障診断は，もっとも難解な問題の一つである．今後とも，これに対する有力な対策が望まれる．

（7） ソフトウェアの信頼性

ソフトウェアの信頼性について，ハードウェアと同様に問題となり，高信頼度でバグのないソフトウェアを実現する手法が探究されよう．特に，障害処理ソフトウェアのデバッグは現在困難であるので，この点に関する改善がほしい．

8. おわりに

以上で，大型計算機の RAS について，現状を中心に解説し，今後の課題についてもふれた．今後ますます重要性を増すであろうこの方向について，読者諸氏の関心を深める一助となれば幸いである．

参考文献

1) Stanga, D. C.: "UNIVAC 1108 Multiprocessor System," SJCC Proceedings, 67-74, 1967.
2) Carter, W. C. et al.: "Design of Serviceability Features for the IBM System/360," IBM Journal of Res. & Dev., Vol. 8, No. 2, 115-126, 1964.
3) Higgins, A.: "Error Recovery Through Pro-

graming," FJCC Proceedings, Vol. 33, 39-43, 1968.
4) Bock, R. V. and Toth, A. P.: "Hardware and Software for Maintenance in the B 5500 Processor," IEEE Int'l. Convention Record, Vol. 13, pt. 3, 21-27, 1965.
5) Kuehn, R. E.: "Computer Redundancy: Design, Performance, and Future," IEEE Trans. on Reliability, Vol. R-18, No. 1, 3-11, 1969.
6) Flehinger, B. J.: "Reliability Improvement Through Redundancy at various System Levels," IBM Journal of R & D, 148-158, April 1968.
7) Bouricius, W. G. et al.: Investigations in the Design of an Automatically Repaired computer," Digest 1st IEEE Computer Conference, September 1967.
8) Chang, H. Y. and Scanlon, I. M.: "Design Principles for Processor Maintenability in Real-Time Systems," FJCC Proceedings, Vol. 35, 319-328, 1969.
9) Carter, W. C. and Bouricius, W. G.: "A Survey of Fault-Tolerant Architecture and Its Evaluation," Computer, Vol. 4, No. 1, January 1971.
10) "International Symposium on Fault-Tolerant Computing," の Proceedings, March 1971.
11) Bartow, E. and Mcguire, R.: "System/360 model 85 microdiagnostics," SJCC Proceedings, 191-197, 1970.

(昭和46年5月13日受付)