オートノミックコンピューティング

日本アイ・ビー・エム（株）東京基礎研究所
福田 剛志
fukudat@jp.ibm.com

自分自身を自律的に管理するコンピュータシステムを作りたい ── コンピュータシステムが社会的基盤となるに従って顕著になった，システムを管理することの難しさを根本から解決するためのグランドチャレンジとして提唱されたのが「オートノミックコンピューティング」である．このオートノミックコンピューティングをテーマとし，昨年 8 月には IEEE 主催の国際会議 Industrial Informatics（INDIN'03）に併設されたワークショップ Autonomic Computing Principles and Architectures（AUCOPA '03）[12] が開かれ，今年 5 月には IEEE 主催の国際会議 International Conference on Autonomic Computing（ICAC '04）[11] の開催が予定されており，注目を集めている．本稿では，オートノミックコンピューティングの問題意識，ビジョン，アーキテクチャ，そして 1 つの研究例を取り上げて，今後取り組むべき技術的な課題を紹介する．

問題意識

コンピュータの進化の歴史は，すさまじい勢いで進められた小型化，高速化，低価格化の歴史（図 -1）であると同時に，応用範囲を拡大していく歴史でもある．応用範囲を拡大するということは，個々のコンピュータにより大きな仕事をさせるということばかりでなく，そのようなコンピュータシステム同士を相互につないで連携させ，統合していくということでもある．インターネットの出現により，この傾向はますます加速されている．この方向性に従って，必然的にコンピュータシステムは「複雑さ」の度合いを急速に増してきた．情報科学の主題の 1 つでもある「組合せ」が，はじめは小さかった複雑さをどんどん増幅させていくためである．その結果として，コンピュータシステムを運用し管理していくことが，非常に多くの労力を必要とする仕事となっている

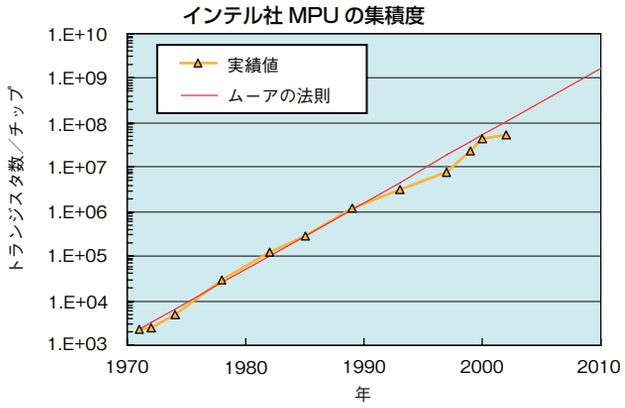

図 -1　インテル社の MPU の実際の集積度と，ムーアの法則「1 つのチップ上に集積できるトランジスタ数はおよそ 2 年で倍増する」[10] をプロットした

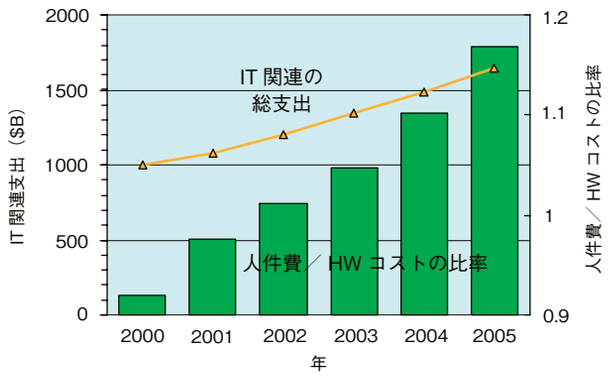

図 -2　全世界の IT 関連総支出および人件費とハードウェアコストの比率（IDC 2001 年 6 月）

図 -3　オートノミックコンピューティングの 4 つの構成要素

(図 -2)．このままいくと人間の管理者の能力の限界を超えてしまい，そのようなシステムを維持することはできなくなるだろう [8]．

本来人間の生産性を高めるために作られたコンピュータシステムを，運用・管理する仕事の生産性がどうも高くない，それどころかコンピュータの進歩の過程で，生産性がどんどん悪化していくようだというのは，なんとも皮肉な話である．それも，コンピュータシステムが発達し，私たちがそれに依存していく過程で必然的に生じた性質である「複雑さ」によって，である．このまま放置すると，いくらすばらしいコンピュータシステムを作ったとしても，管理コストの増大で，トータルの生産性はまったく向上しないというようなことにもなりかねない．ある研究 [14),13)] によると，インターネット上のサービスが停止に追い込まれるような障害の原因のうち，約 40% はオペレータ（人間）のエラーで，他の原因を抑えてトップであるという．

この問題は，私たちがこれまで最重要の課題だと考えてきたコストパフォーマンスをいくら追求しても，解決される性質のものではない．個々のコンポーネントの品質を上げたり，システム管理ソフトウェアの使い勝手をよくすれば解決できるような問題でもない．なぜならこの問題は，あらかじめ予想されていなかったさまざま要因，たとえばシステム同士の接続・統合や，ワークロードの性質の変化や，時には社会制度の変化などによって生じているためである．この「複雑さ」に対処するためには，まったく新しいアプローチが必要となる．

オートノミックコンピューティング

人間は，無意識のうちに，気温が高くなると汗をかき，運動をすれば心臓の鼓動が速くなり，明るくなれば瞳孔が小さくなる．実際人間は，現在のコンピュータシステムがおよびもつかないほど複雑だが，オートノミック（自律的）にシステムが維持されている．つまり，私たちの体は無意識のうちに，環境に適応するように管理･調整されているのだ．これと同じように，コンピュータシステムも内外の環境に自己を適応させる自律神経系（autonomic nervous system）を持つシステムが実現できたらどうだろう．そして運用管理者が意識しなくても，すべてを自動的に最適化できる仕組みが提供できたら．2001 年，私たちはこのような夢を描いて，世界の産業界，学術界に，「複雑さ」の問題を克服するための協力を呼びかけた [6]．それが「オートノミックコンピューティング（Autonomic Computing）」である．

これまで，生物の仕組みをまねた情報処理技術はいくつか提案されている．たとえばニューラルネットワーク，遺伝的アルゴリズム，セルオートマトンは生物をモデルにしているといえるだろう．これらはどれも計算方法に主眼が置かれていたが，オートノミックコンピューティングは，結果として自律的に動作しているという生物の基本的性質をコンピュータシステムにも取り入れていきたいという，技術開発の方向性である．

オートノミック（自律的）なシステムが持つべき特性とはどのようなものがあるだろう．次の 4 つの観点の「自己管理能力」が考えられている（図 -3 参照）．

・自己構成

人間に与えられたポリシーに基づいて，環境の変化やコンピュータシステム内の変更にダイナミックに対応する能力．これらの変更には構成の新規追加，不要となった構成撤去，ワークロードの劇的な増減への対応などがあり，自分自身の構成はもちろん，システム内の関連する他の製品に対する変更も含め，システム全体を再構成することが求められる．

・自己修復

障害の原因となる可能性のあるものや重要なイベント

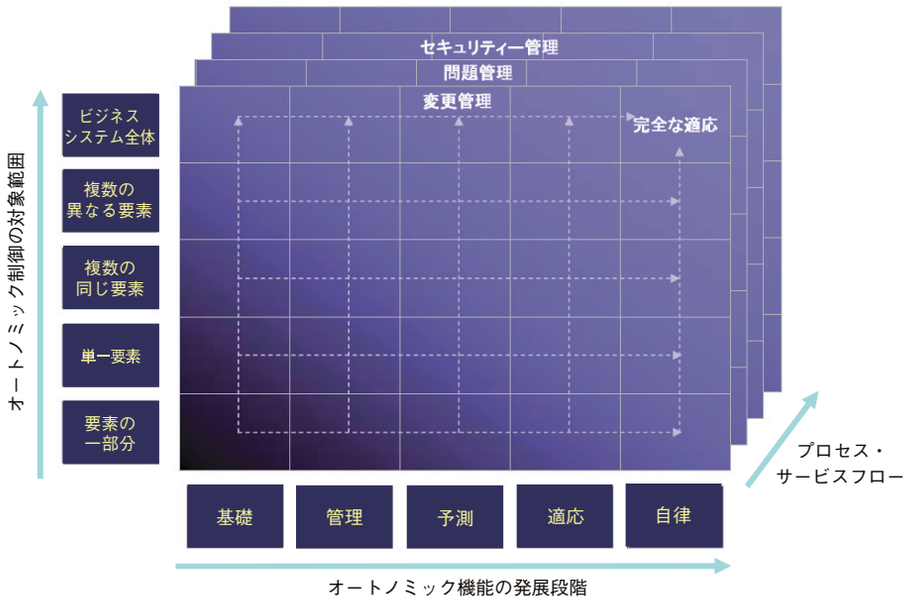

図 -4　オートノミックの発展・展開モデル

を発見し，重大な障害となる前に，システムを中断させることなく適切なアクションをとり修復する能力．

・自己最適化

エキスパートの介在なしでユーザやビジネスのニーズに応じて自分自身を調整し，最適化し続ける能力．IT レベルの効率の追求ではなく，ビジネスレベルでの効用（たとえば利益）を最大化することが求められる．

・自己防御

敵意ある行為や侵入者を発見し，攻撃から身を守り，より攻撃されにくくするためのアクションをとる能力．

これら 4 つの自己管理能力は互いに独立したものではない．構成変更の誘因はそれぞれの切り口で異なるが，構成をダイナミックに変更できるためには，これらが協調して働く必要があるだろう．

これはいわゆる普通の「自動化」と何が違うのだろうか．

比較的単純な作業の自動化は，コンピュータが最も得意とする分野である．したがってコンピュータの管理も，単に単純作業を自動化していけばよいだけではないか，と思われるかもしれない．もちろんそのような自動化はある程度の助けにはなるだろう．しかしそれは，決まりきった作業を事前にプログラムしておいて，コンピュータに行わせるということであり，運用時（オペレーション時）のエラーを設計時あるいはプログラミング時のエラー（バグ）にすり替えるだけにもなりかねない．また，より本質的な事実として，単純で部分的な自動化が済んだ後に人間に残されるのは，本当に難しい仕事になるということである．人間のオペレータは，部分的な自動化が覆い隠してしまう不完全な（時には間違った）情報をもとに，ミスをしてはいけないという大きなプレッシャーの中で，難しい選択を迫られることになる．つまり，これまでのような安易でアドホックな自動化は，システムの複雑さを増し，かえってシステム管理を困難にしてしまう可能性すらあるのだ．

したがって，私たちが目指すオートノミックコンピューティングでは，抽象度のレベルを上げ，より高いところから全体を見渡すことで，運用・管理のレベルを質的に向上させる必要がある．

発展・展開モデル

オートノミックコンピューティングは将来に向けた大きな目標であり，今すぐに実現できるものではない．たとえ技術的に実現できたとしても，現在の資産をすべて捨てて，革命的にオートノミックなシステムに乗り換えるというのは現実的ではないだろう．

そこで私たちは，オートノミックコンピューティングの発展段階を「基礎」「管理」「予測」「適応」「自律」の 5 段階に分け，オートノミックに向けて段階的に前進していくことを提案している（図 -4 の横軸）．「管理」レベルは「監視」の自動化，「予測」レベルは「監視」と「分析」の自動化，「適応」レベルは「監視」「分析」「計画」「実

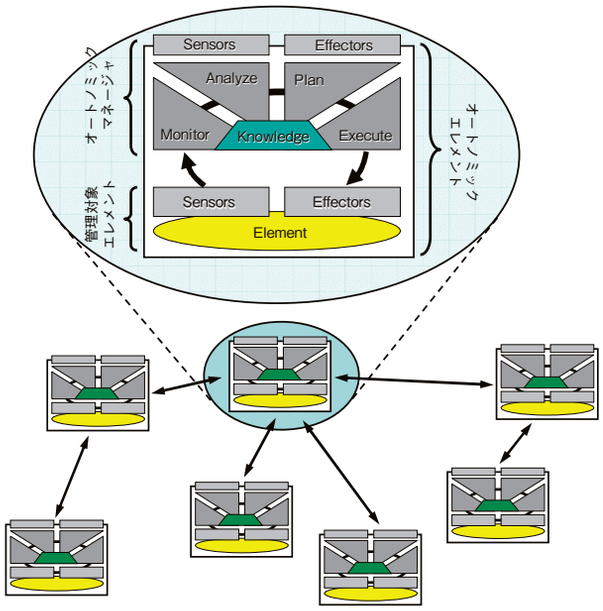

図 -5　オートノミックエレメントの内部構造．エレメントはオートノミックマネージャを通じて，人間や他のエレメントと対話する

行」の IT レベルのポリシーに基づく自動化，そして「自律」レベルでビジネス・ポリシーに基づく自律管理が完成する．

たとえばハードディスクドライブのレベルの自律性と，それが集まったストレージシステムの自律性，さらにそれを使うデータベースサーバの自律性，もっと上位のアプリケーションとしての自律性といったように，オートノミック化するシステムの範囲によっても，さまざまな段階が考えられる（図 -4 の縦軸）．また，コンピュータシステムの運用管理プロセスにはさまざまな項目があり得る（図 -4 の奥行き方向）．これらの観点からも，より広いスコープでの自律性へと段階的に発展していくべきだろう．

このようにオートノミックの発展過程を整理しておくと，コンポーネントやシステムの現状を理解・評価し，今後進むべき方向を知るための基準として用いることができる．たとえば，文献 9）は製品のオートノミックコンピューティングの成熟度をベンチマークしようという試みである．ディペンダビリティベンチマーク [1] は，製品を組み合わせて作るシステムのディペンダビリティを評価する．また，コンピュータシステムだけでなく，それを運用・管理・利用する組織やビジネスプロセスを含めた企業活動全体を評価するということも行われている [5), 4)]．

アーキテクチャ

すでに述べたように，アドホックな自動化では対処できないのだとすると，どのようにすればオートノミックなシステムを構築することができるのだろう．

各コンポーネントが自律的に動作するためには，まず「自分自身を知る」そして「自分の周りの状況を知る」といった，情報を収集するためのセンサ機能を持つことが必要となる．そして，判断した結果に基づいてコンポーネントの状態を変化させるためのエフェクタ機能も必要である．何万種類もあるコンポーネントが独自のインタフェースを持つセンサとエフェクタを持ったとすると，それが集まった全体を自律的に制御する仕組みなど作れるはずがない．したがって，そのインタフェースはオープンな標準に従って決めなければならない．また「自己管理する」とはどういうことだろうか．私たちはこのようなことを整理して，アーキテクチャのブループリントとして発表した [7)]．以下，その概要を紹介する．

オートノミックシステム（autonomic system）は，相互に影響を及ぼし合うオートノミックエレメント（autonomic element）の集合である．オートノミックエレメントは，何らかの資源を持ち，人間や他のオートノミックエレメントに対してサービスを提供する，システムの構成要素である．オートノミックエレメントは，自分自身の振舞いと他のオートノミックエレメントとの関係を，人間や他のエレメントが設定したポリシーに従って，自律的に管理する．システムの自己管理能力というものは，個々のオートノミックエレメントの自己管理能力だけでなく，オートノミックエレメント間の相互作用によって生じるものである．これは，たくさんの蟻の個体同士が互いに情報を交換し影響を及ぼし合うことによって，群れ全体として知的に振る舞うというのによく似ている．

図 -5 はオートノミックエレメントの内部構造である．オートノミックエレメントは，いくつかの管理対象エレメント（managed element）と，それらを制御し代表する 1 つのオートノミックマネージャからなる．管理対象エレメントは，今日普通のシステムの構成要素となんら変わりはないが，オートノミックマネージャによってモニタされ，制御され得るように，すでに述べたようなセンサとエフェクタを備えている．管理対象エレメントは，CPU，ハードディスク，プリンタのようなハードウェア資源でもよいし，データベースなどのソフトウェア資源でも構わない．最も上位のレベルでは，管理対象エレメントはアプリケーションサービスか，あるいは何か

1つの事業そのものであるかもしれない．オートノミックマネージャは，それが管理する管理対象エレメントと外側の環境を「モニタ（monitor）」し，情報を「分析（analyze）」した結果に基づいて「計画（plan）」を立て，それを「実行（execute）」することによって，人間の管理者が果たしてきた責任を肩代わりする．

オートノミックコンピューティングが段階的に発展していく過程では，既存の管理対象エレメントに対して，徐々により洗練された機能を持ったオートノミックマネージャが追加されることにより，システムのオートノミックレベルが高まり，対象範囲が広がっていくことになるだろう．また将来は，オートノミックマネージャの機能が管理対象エレメントの基本機能に統合されていくことにより，オートノミックマネージャと管理対象エレメントの区別は薄れていくかもしれない．

オートノミックエレメントにも，その外側に向けてセンサとエフェクタがあることに注意してほしい．このようにしておくことで，オートノミックエレメントも1つの管理対象エレメントとして，上位のオートノミックマネージャの管理対象とすることができる．このように，オートノミックエレメントがビルディングブロックとなって，より大きな，抽象度の高いオートノミックシステムを構築することができる．

低いレベルにおいては，オートノミックエレメントの「行動範囲」はあまり大きくなく，一部はハードコードされた単純な自動化かもしれない．特に個々のコンポーネントのレベルにおいては，フォールトトレラントの分野で培われた技術を用いて，壊れないエレメントにするというのは自然な考えである．高いレベルにおいては，従来固定的だったエレメントの振舞いは，たとえば「この効用関数を最大化せよ」といったような，上位のマネージャから与えられるゴールを達成するために動的に決定されるようになる．エレメント間の接続関係も，固定的だったものから，エレメント同士が交渉することによってダイナミックに確立されるものになる．

オートノミック自己修復システム

研究開発の例として，自己修復能力を持つコンピュータシステムの実現に向けた私たちの取組みを紹介する．

オートノミックにコンピュータシステムの障害を検知し，原因を突き止め，さらにその原因を自動的に取り除き，解決してしまうようなシステムを作りたい．そのためには，まずシステムを構成する各コンポーネントの状態をモニタする仕組みが必要である．現在多くのコンポーネントは，状態変化を独自のフォーマットでイベントとして出力し，多くの場合ログデータとして記録している．従来のイベントデータは，それを出力するコンポーネントのことだけを考えてデザインされているため，そのコンポーネントの問題判別に用いることはできても，システム全体の問題を解決することには不都合が多い．たとえば，あるコンポーネントがサービスを開始したというイベントは「コンポーネントが実行を開始した」と表現されるかもしれないし「コンポーネントが起動した」と表現されるかもしれない．人間は，これらが同様の状況を表現していることをすぐ理解できるが，こんな些細なことでも，コンピュータは簡単には理解できない．特定の種類のコンポーネントに限定せず，一般的にシステムの状況を解釈する仕組みを構築するためには，システムで発生しているイベントを表現する共通の「言語」を決めておく必要がある．

現在，オープンソースコミュニティのEclipse Hyades[2]プロジェクトで用いられているCommon Base Event（CBE）というフォーマットがOASIS（Organization for Advancement of Structured Information Standards）に提案されており，私たちはこれを共通言語として使用することにしている[3]．CBEとは（状況を監視しているコンポーネント，影響を受けるコンポーネント，状況データ）の3つ組みで，「状況を監視しているコンポーネント」が観測した「影響を受けるコンポーネント」の状況を定められた語彙で表現した「状況データ」として報告するものである．このように統一されたフォーマットを採用することで，従来のログに存在した微妙な表現の違いが吸収され，その後の分析プロセスが大幅に単純になる．まだCBEに対応していないコンポーネントに対しては，簡単な変換ルールを与えることで，従来使われている固有のデータフォーマットをCBEに変換するアダプタを用いる．

図-6は私たちが目指すシステムの概観である．各コンポーネントはその状況をイベントとしてCBEフォーマットで出力する．オートノミックマネージャは，障害が発生したとき，各コンポーネントから収集されるイベントを元に，状況の分析を行って原因を突き止め，解決策の計画を立て，それを実行する．コンポーネント内で診断ができるような障害は，そのレベルの分析エンジンが診断を下し，必要な修正アクションをとる．他のコンポーネントの症状と関係付けた上で診断が必要な障害は，上位レベルの分析エンジンが判断を下すといった，階層的な診断が行われる．このほかに，ネットワークを流れるデータを監視することによりシステム全体の状況

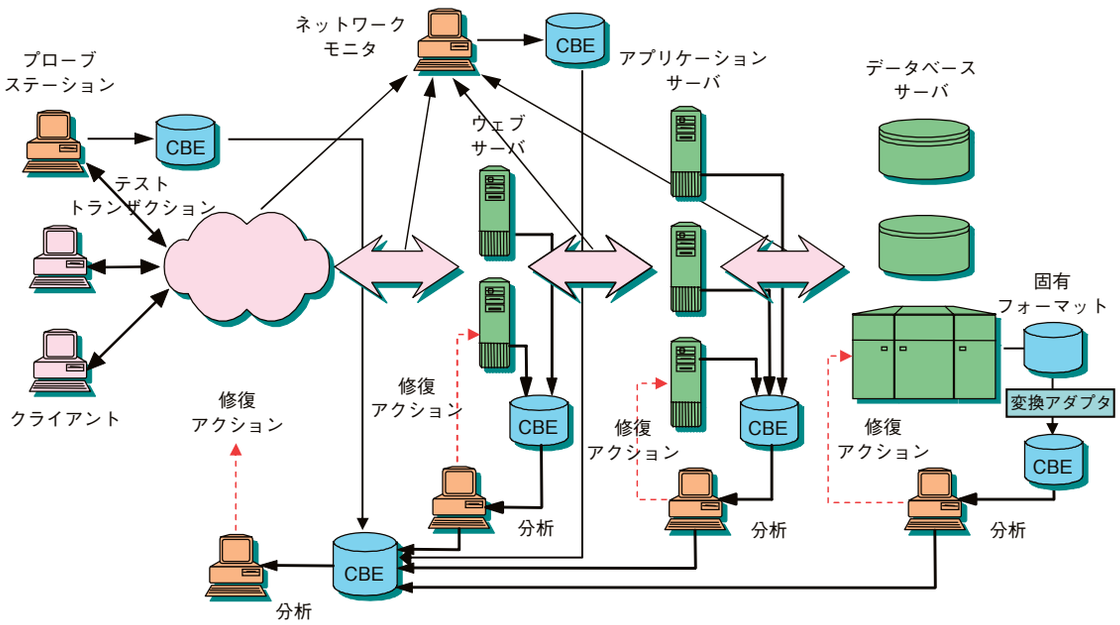

図 -6　自己修復システム

を把握する，ネットワークモニタを用いた分析エンジンや，積極的にテストトランザクションを発生させ，その実行過程を観察することによりシステムの健康状態を把握する，プローブステーションを用いた分析エンジンも組み合わせて使用する．これらの分析エンジンは互いに協調して動作し，障害の場所と原因を特定する．原因判定は，対象とする障害の範囲をごく小さく絞れば，比較的容易に実現することができるが，その範囲をどうやって拡大し，実用的にしていくかが課題である．そのためには，実際に運用されているシステムで生じる障害とその原因究明の経験を取り込んでいくことが欠かせない．

原因が特定できたなら，引き続いてそれを解決するための一連のアクションを計画し，さらには自動的に実行する．しかし，実際のシステムに自動的な修復アクションを適用してしまうには，まだリスクが大きすぎる．当面は管理者に対して解決策をアドバイスとして提案するというのが現実的であろう．

おわりに

オートノミックコンピューティングは，まだ第一歩を踏み出したばかりである．これまで分散コンピューティング，フォールトトレランス，ディペンダビリティ，ソフトウェア品質工学，エージェント技術など多くの分野で培われた技術を総合していくだけではなく，今後より多くの研究者・技術者がこの分野で活発に活動し，技術革新が進んでいくことを期待している．

参考文献
1）Dependability Benchmark Project; http://www.laas.fr/DBench/
2）Eclipse Hyades Project; http://www.eclipse.org/hyades/
3）Automating Problem Determination: A First Step Toward Self-healing Computing Systems; http://www.ibm.com/autonomic/pdfs/Problem_Determination_WP_Final_100703.pdf
4）Autonomic Computing Readiness Assessment; http://www.ibm.com/jp/autonomic/solutions/consulting.shtml
5）Resilient Business and Infrastructure Solutions; http://www.ibm.com/services/strategy/files2/G510-3124-01.pdf
6）Autonomic Computing: IBM's Perspective on the State of Information Technology; http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf（Oct. 2001）.
7）An Architectural Blueprint for Autonomic Computing; http://www.ibm.com/autonomic/pdfs/ACwpFinal.pdf（Apr. 2003）.
8）Kephart, J. O. and Chess, D. M.: The Vision of Autonomic Computing, IEEE Computer, pp.41-50（Jan. 2003）.
9）Lightstone, S.: Towards Benchmarking Autonomic Computing Maturity, IEEE Workshop on Autonomic Computing Principles and Architectures（AUCOPA '03）（Aug. 2003）.
10）Moore, G. E.: Cramming More Components onto Integrated Circuits; ftp://download.intel.com/research/silicon/moorespaper.pdf, Electronics, 38（8）（Apr. 1965）.
11）IEEE International Conference on Autonomic Computing（ICAC '04）; http://www.autonomic-conference.org/
12）IEEE Workshop on Autonomic Computing Principles and Architectures（AUCOPA '03）; http://www.enel.ucalgary.ca/INDIN03/AutonomousWS.html
13）Oppenheimer, D., Ganapathi, A. and Patterson, D. A.: Why Do Internet Services Fail, and What Can Be Done About It ?, In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems（USITS '03）（Mar. 2003）.
14）Patterson, D. A.: Availability and Maintainability » Performance: New Focus for a New Century, In USENIX Conference on File and Storage Technologies（FAST '02）, Keynote Address（Jan. 2002）.

（平成 16 年 3 月 15 日受付）