論文

〔知働化研究会誌第２号〕
知働化研究会
Executable Knowledge and Texture Laboratory

分散システムが決める進化の方向性

萩原正義（はぎわらまさよし）

Microsoft Corporation

クラウドによるサービス提供 58
クラウドの分散システムの動作 58
分散システムの設計上の困難な課題 60
分散システムの誤謬 61
サイエンスと工学の役割[5] 62
イノベーション、ソフトウェア開発の複雑さのために 63
参考文献 64

クラウドによるサービス提供

現在のような成熟社会では、様々なオンラインサービスの成功は提供してみてからでないとわかりません。サービスがユーザにとって価値があり、使い勝手がよいものになるためには、変更しながら改善できること、成功の見込みがないものは早期に撤退して再試行することが求められます。また要求の変化や需要の喚起のために迅速な対応を取れなければなりません。サービスの利用が急激に増大した場合にも機会損失することなく、品質を保証しなければなりません。

これらのサービスの要求に対応するにはもはやクラウドコンピューティングは不可欠です。サービスの試行、早期立ち上げ、迅速な変更、急激な需要への対応、早期の撤退などに必要な機能を提供するからです。だだし、クラウドだけですべてを実現するかどうかは、品質、コスト、ノウハウなどの制約を考慮しなければなりません。場合によってはオンプレミスのシステムとのハイブリッドが現実的かもしれません。

クラウドの特長の１つに、ソフトウェアによるリソース管理によって、実行時のリソース（コンピュート、ストレージ、ネットワーク）の容量を増減し、構成定義を変更できる機能があります。Elasticity（柔軟性）と呼ばれる機能です。また、実行時にアプリケーションコードの変更が可能です。これは、クラウドが分散システムによって複数サーバで動作するために、アプリケーションコードの配布をサーバ毎に行うことで、運用中の配布が実行可能なためです。また、分散システムの別のメリットは、全体が１つの仮想サーバのように動作するため、部分的なサーバの障害やネットワークの障害の影響を限定化できることです。これにより、サーバやネットワークのハードウェアに要求される信頼性を下げることで全体のコストを下げ、運用コストに競争力を持たせます。また、サーバやネットワークのハードウェアの容量（スループットや遅延時間）の限界を超えるために、複数のそれを束ねることで線形に近い性能を引き出せるようにします。このスケールアウトも分散システムを実現するソフトウェア技術によってもたらされます。

クラウドの分散システムの動作

クラウドを支える分散システムの役割を理解したところで、もう少し詳しくその動作を見ていきましょう。図１はクラウドの分散システムの動作を示しています。順を追って説明しましょう。

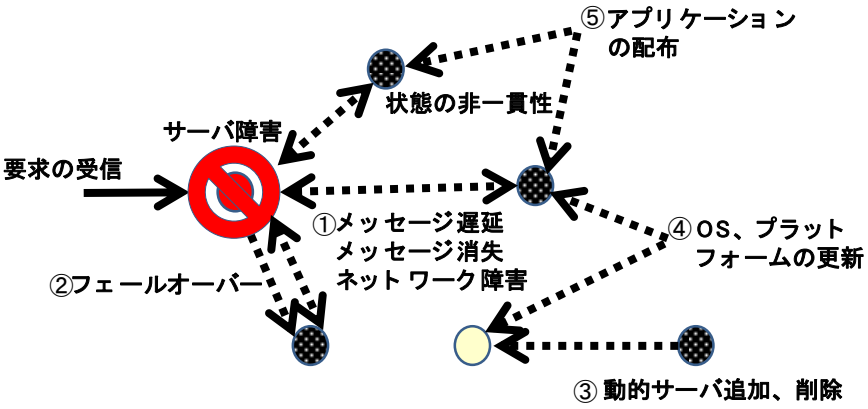

図１　分散システムの動作

（１）　ユーザからの要求メッセージがインターネット経由でクラウドのサーバ（図の左手にある入口のサーバでリーダーと呼ぶ）で受信されます。受信されたメッセージはこの１台のサーバからクラウドの複数（通常３台以上）のサーバへ複製されます。複製は耐障害性を与えるために必要です。ただし、複製へのメッセージ送信ではメッセージ遅延、消失が伴います。また一部の複製にはメッセージ遅延、ネットワーク障害などにより届かないことがあるので、そのサーバ状態は同一にはなりません（状態の非一貫性）。

（２）　要求の処理中にリーダーがサーバ障害を起こした場合、複数の複製のうちの１台のサーバをリーダーに昇格し、リーダーの役割を引き継がせます。リーダーの切り替えをフェールオーバーと言います。ただし、フェールオーバーでもメッセージ遅延やネットワーク障害の発生を想定しなければなりません。フェールオーバーが完了するまではサービスは処理できません。

（３）　ユーザの要求の負荷に応じて、クラウドの分散システムの elasticity の特長により、サーバの追加や削除が実行されます。これによりリソースの効率的運用が可能となります。

（４）　分散システムの OS やミドルウェアなどのプラットフォームはメンテナンスにより運用中に更新されます。

（５）　分散システムで動作するサービスのアプリケーションコードが運用中に配

布されます[3]。サービス更新が常時可能なことでサービス改善のスピードが上がり、ユーザの要求に迅速に対応可能となります。

分散システムの設計上の困難な課題

次に、クラウドの分散システムに期待される動作を実現する上で、設計上の困難な課題の例をいくつか見ていきましょう。

(a) 分散システムがそれぞれのサーバのハードウェア、ソフトウェアの限界を超えるために、複数のそれを束ねることで線形に近い性能を引き出させる例[1]：
複数のサーバを同時に動作させることでサービスの処理性能を上げるには、それぞれのサーバに担当範囲を分担させて、同時実行させる方法を取ります。その代表的な方法に「シャーディング」があります。シャーディングとはそれぞれのサーバのデータ操作の担当範囲に合わせてデータを分割する方法です。たとえば、ユーザ ID やユーザ名のまとまりで分割します。100 万人の登録ユーザを 10 万人毎に 10 台のサーバに ID で分割（たとえば、下一桁の数字で割り振る）したり、ユーザ名の姓の頭文字で分担（ア行、カ行など）させたりします。ただしシャーディングを使ったとしても、一時的に特定サーバに要求が偏ることがあり、そのサーバが分散システム全体の動作を遅延または停止させてしまう問題が発生します。この場合、サーバを追加して負荷分散させようとしても、うまくユーザの要求の一部が負荷の集中したサーバから追加のサーバに逃がせるかは、分散システムの複雑な振る舞いによって予想できません。なぜなら、シャーディングのやり直し、やり直しに伴うインデックス/メタデータや構成変更、リソースの削除などが含まれているからです。

(b) Elasticity への追随の例[2]：
ユーザの要求の負荷変動によってメッセージの待ち行列の長さや CPU に使用率などが一定以上になると自動的にサーバやアプリケーション数を増やす機能をオートスケールと言います。オートスケールによって負荷に応じた、応答時間の調整、効率的なリソースの利用が可能となります。ただし、オートスケールは急激な負荷の変動に対しては追随が遅れます。したがって、リソースが不足して、ユーザの要求のサービスができない問題が発生します。負荷の変動に迅速に対応するためには、より精度の高いソフトウェアの開発が必要で、既存のクラウドの機能では不十分となります。

(c) アプリケーションの配布とオートスケールの干渉の例[2] [3]：
アプリケーションの配布とオートスケールによるサーバの追加や削除が同時に発生した場合、追加のサーバにアプリケーションが配布されない、サーバ毎のアプリケーションのバージョンが異なる不整合を発生させます。したがって、アプリケーションの配布中はオートスケールを発生させないこと、オートスケール中はアプリケーションの配布を禁止することが必要となります。いずれも、ユーザの要求の負荷変動への不十分な対応、迅速なサービスの改善への対応の遅延を招くので、サービスの品質を下げてしまいます。このため、できるだけ禁止の期間、回数を減らすための計画や運用が必要となります。

分散システムの誤謬

これまで分散システムには、Fallacies of Distributed Computing Explained[4]によれば、以下の仮定が成立すると信じられてきました。

1. The network is reliable. ネットワークは信頼できる
2. Latency is zero. 遅延はゼロ
3. Bandwidth is infinite. バンド幅は無限大
4. The network is secure. ネットワークのセキュリティは保護される
5. Topology doesn't change. ネットワーク接続構成には変更がない
6. There is one administrator. 管理者はひとりだけ
7. Transport cost is zero. 転送コストはゼロ
8. The network is homogeneous. ネットワークの構成要素は同機種

しかし、これらはどれもが成立しないことが明らかになってきました。なぜなら、クラウドの分散システムではこれらが成立しないことが課題として支配的になってきているからです。

より科学的アプローチが求められていると言い換えられます。サービス提供のためにアプリケーションを開発するとき、どういう理由で特定の技術を選択したかの説明責任が求められます。その理由づけにおける科学的根拠の比重が増えたと言えます。

サイエンスと工学の役割[5]

図２はコンピュータサイエンスとソフトウェア工学の役割を示しています。コンピュータサイエンスはハードウェアやソフトウェアの物理法則、原理（理論）を扱います。一方、ソフトウェア工学はソフトウェア開発における決まり、規律、（ベスト）プラクティスを扱います。たとえば、アルゴリズム、リソース管理、プロトコル設計などはコンピュータサイエンスの分野で、工数の見積もり、プロジェクト管理、デザインパターンなどはソフトウェア工学の分野です。

コンピュータサイエンスの原理や理論の実現にも、ソフトウェア工学のプロジェクトの実装にも共通してプログラミングが伴います。また、いずれの分野にも制約、トレードオフが存在します。

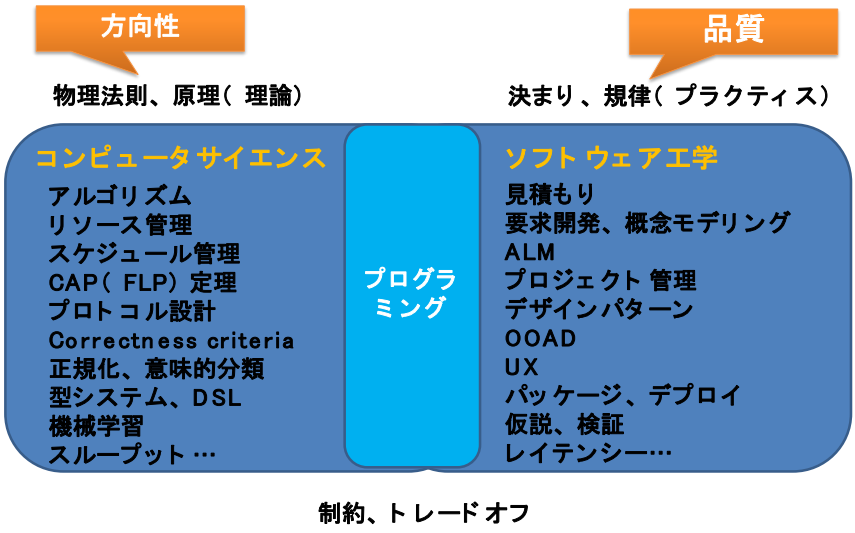

図２ コンピュータサイエンスとソフトウェア工学の役割

これまでは、オブジェクト指向設計（OOAD： Object Oriented Analysis and Design）、デザインパターン、要求開発、ALM（Application Lifecycle Management）といったソフトウェア工学によってどのようにソフトウェアの品質を高めていくかという問題が注目されてきました。これに加えて、クラウドの分散システムによる開発の本格化で、アルゴリズムの選択、その正しさを検証する条件（Correctness criteria）、分散システムの制約（CAP 定理）などの重要性が高まっています。

イノベーション、ソフトウェア開発の複雑さのために

今後のイノベーションを起こすには、どう正しく作るかのソフトウェア工学ではなく、実現可能性を持った何を作るかのコンピュータサイエンスにかかっていると思われます[6]。その点では、コンピュータサイエンスはこれからの方向性を決めると言えるでしょう。私自身もこれまでソフトウェア工学を担当した経験が長かったのですが、クラウドの誕生をきっかけにコンピュータサイエンスに軸足を移しました。
最後に、ソフトウェア開発の複雑さにいかに対応すべきかを考えてみましょう。ソフトウェア開発の複雑さには、以下の 4 種類があると考えられます。

1. 問題領域の複雑さ
2. 解決領域の複雑さ
3. ビジネス要求、機能要求の複雑さ
4. ソフトウェア開発の非機能要求の複雑さ

1 はソフトウェア開発の対象となる領域、原子力発電、航空機の運航管理など、2 はソフトウェア開発で利用する技術の領域、分散システム、データベース、言語システムなど、3 はソフトウェア開発の対象のビジネスの制約、コスト、ステークホルダの利害関係など、4 はソフトウェア開発の品質に関わるデータ項目数などの規模、可用性と応答時間の関係、障害モデルなど、の領域です。

このうち、コンピュータサイエンスが関わるのは 2 と 4 の一部です。したがって、依然としてソフトウェア開発の多くはソフトウェア工学が対象する領域です。コンピュータサイエンスが技術の方向性を決めつつ、その中で選択した技術でどう開発し実現させるかはソフトウェア工学の範疇です。それぞれがその役割を担いつつ、発展していくと考えられます。

さて、読者のソフトウェア開発に携わる皆さんはこれからどちらに軸足を置いていくつもりですか？

参考文献

1. “Foursquare's MongoDB Outage,” http://www.infoq.com/news/2010/10/4square_mongodb_outage
2. “クックパッドのデプロイとオートスケール,” http://www.publickey1.jp/blog/14/110jaws_day_2014_1.html
3. “グリーが OpenStack で目指す"適材適所"なクラウド環境,” http://www.atmarkit.co.jp/ait/articles/1403/05/news021.html
4. Arnon Rotem-Gal-Oz: “Fallacies of Distributed Computing Explained.”
5. “クラウド時代のソフトウェア技術者にはコンピュータサイエンスこそ武器になる,” http://codezine.jp/article/detail/7665
6. クリストファー・スタイナー: “アルゴリズムが世界を支配する.”