〈クラウドの技術課題，将来展望〉

5

スケールアウトの技術

首藤 一幸　東京工業大学

クラウドのソフトウェアには，コンピュータの台数に応じた性能を引き出すこと，つまりスケールアウトが強く求められる．本稿ではスケールアウトの手法を解説し，データストアや，データストアから取得したデータをキャッシュするソフトウェアへの応用を紹介する．

分散データストア

データストアへのアクセス性能や容量を稼ぐためには，次の方策が考えられる．

- スケールアップ／vertical scaling
  サーバの性能を強化する．
- スケールアウト／ horizontal scaling
  サーバの台数を増やす．

スケールアップではすぐに限界に達するし，何よりも，ハードウェアが高くつく．2 倍の性能・容量を得るために 10 倍の金額が必要となるといった具合である．そこで，クラウドの構築，またはそこまで行かずとも，1 台では不足という場合，スケールアウトを目指すこととなる．

アクセス性能を稼ぐためには，全データを別のサーバに複製するという方法がある．複製元，つまりマスタへの書き込みを，別サーバ上の複製にコピーする．これによって読み出しアクセスを複数台に分散させることが可能となり，読み出しの性能を稼げる．しかし，台数を増やしても容量は稼げず，また，複製を増やすに従って，一貫性の維持，つまり複製の更新がコスト高になっていく．

より多くのサーバを活用したい場合や容量を稼ぎたい場合，データの分割が必要となる．リレーショナルデータベース（RDB）の場合，表ごとに異なるサーバに持たせる方法，表の垂直分割，水平分割という選択肢がある（図 -1）．垂直分割とは 1 つの行を複数の表に分割することであり，通常，表の正規化やセキュリティ等止むを得ない理由で行われる．また，分割後の表の数もたかが知れている．そのため，以下では水平分割，つまり sharding について考える．

データ（各行）を各サーバにどう分担させるか？　データストアへの問合せ内容から担当サーバが決まることが重要である．さもないと，問合せのたびに全サーバに問い合わせることとなり，アクセス性能を稼ぐことはかなわない．そのためには，データベース設計者が特定の列を分割キー（partition key）として指定し，データベース管理システムはその列の値に従って各行の担当サーバを決める，ということを行う．

これは，リレーショナルデータベースではなく key-value ストアでも同様である．key-value ストアとは，キーにデータ（値）を結びつけて格納し，キーを指定することでデータを検索できるデータストアである．分散 key-value ストアでは，与えられたキーに従って担当サーバを決める．

キーから担当サーバを決める規則は，いろいろ考えられる．たとえば，対応表をあらかじめ用意しておく方法や，キーの値の範囲に応じて担当サーバを決める方法などがある．後者はたとえば，顧客 ID が 0 から 9999 までは 1 番サーバといった方法である．これらの方法は，やってくるキーやキーごとの頻度があらかじめ分かっている場合はよいが，そうでない場合はうまく働かない．たとえば Web コンテンツの URL がキーとして次々やってくる場合を考えると，対応表や範囲指定では，サーバ群への負荷分散，つまり担当するキーの数を均等に保つことは難しい．

ここでよく行われるのが，キーのハッシュ値に基づく担当サーバ決めである（図 -2）．サーバが 4 台なら，キーに対して 0 から 3 までの 4 通りの値を返すハッシュ

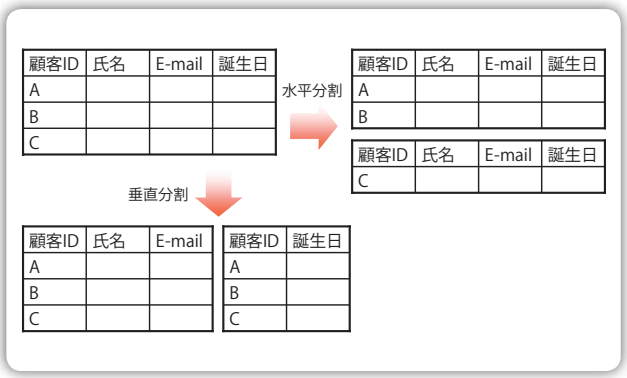


図 -1　リレーショナルデータベースでのデータ分割

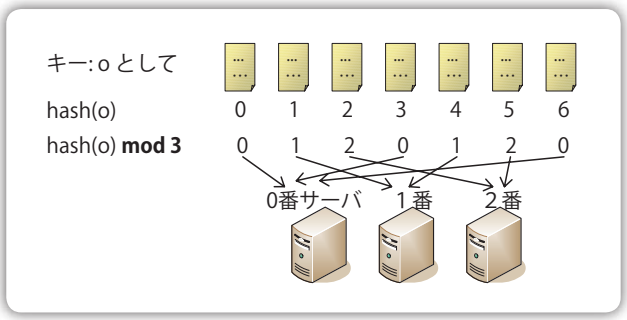


図 -2　ハッシュ関数を用いた担当サーバ決め

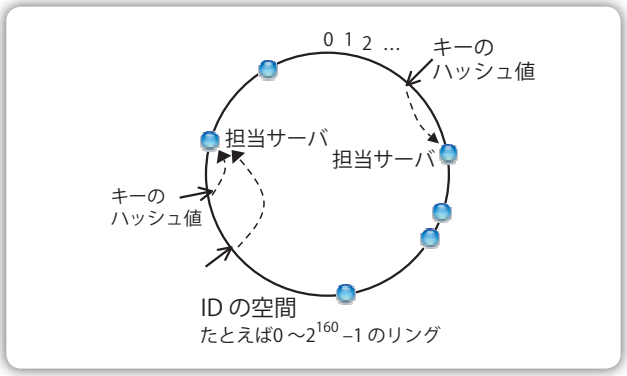


図 -3　コンシステントハッシングの条件を満たす
担当サーバ決め手法の例（時計回り方式）

関数を用意する．キーに対するハッシュ値が 1 番であれば 1 番サーバ，というように担当サーバを決める．こういったハッシュ関数の例を挙げる．

担当サーバ番号 = H(キー) mod サーバの台数

ここで H としては暗号学的ハッシュ関数，SHA-1 や MD5 などを使える．ただし，ここで使うハッシュ関数に一方向性は不要なので，暗号学的ハッシュ関数である必要は特にない．

コンシステントハッシング

ハッシュ値に基づく前述の方法を用いると，未知のキーに対してもサーバごとの担当キー数を均等にできる．しかしこの方法にも，サーバ数の増減には弱いという弱点がある．

性能や容量を増強するためにサーバを増やす，または，減らすという状況を考える．ハッシュ値に基づく方法では，サーバの台数が変わると，ほとんどすべてのキーに対する担当サーバが変わってしまう．たとえば関数 *H* が返す値が 1234 だった場合，台数が 3 から 4 に増えると，担当サーバは *1234 mod 3 = 1* から *1234 mod 4 = 2* に変わる．サーバが *N* 台から *N+1* 台に増えた場合，全キーの *N/(N+1)* について担当サーバが変わってしまう．サーバを 9 台から 10 台に増やすと全キーの 90% について担当サーバが変わる計算である．担当サーバが変わったデータをサーバ間で再配分することは，データが多い場合，ほとんど不可能である．

また，サーバの台数が増えるに従って故障は日常茶飯事となり，増減は日常的に起こるもの，という前提を置かざるを得なくなる．平均故障間隔（MTBF）が 15,000 時間，つまり 1 年 8 ～ 9 カ月というサーバがあったとして，その 1 台が 24 時間稼働し続ける確率は 99.8% である．しかし，10 台となると，わずか 44 日間で，故障が起きない確率は 49.5% と 5 割を切る．

そこで，サーバの増減に耐える担当サーバ決めの方法が考え出された．ある種の手順で担当サーバを決めると，台数の増減があった場合でも，台数 *N* として，担当サーバが変わるのは全キーの *1/N* 程度で済む．この性質を満たす担当サーバ決め手法を**コンシステントハッシング**と言う．これはもともと，複数のサーバで Web のキャッシュサーバを構成するという状況を想定して考え出され，1997 年に論文が発表された [1), 2)]．

この条件を満たす担当サーバ決めの方法は，たとえばこういうものである．0 ～ 2^{160}–1 の整数値をとる ID の空間を考える．これは暗号学的ハッシュ関数 SHA-1 が返す値の範囲である．**図 -3** の通り，最大値 2^{160}–1 の次の値は 0 となるリングを想定する．各サーバに 0 ～ 2^{160}–1 の範囲から ID を割り当てる．キーにも同様に ID を割り当てる．具体的には，たとえば SHA-1 の計算結果を ID とすればよい．図 -3 の通り，キーの ID から ID 空間を時計回りにたどり，最初に行き当たったサーバを担当サーバとする．

この手順に従って担当サーバを決めると，サーバの増減時にも，キーの *1/N* 程度しか担当サーバは変わらない．つまり，コンシステントハッシングの条件を満たす．**図 -4** の通り，3 台サーバ A，B，C があったところにサーバ D を追加する状況を考える．すると，サーバ B が担当していた範囲の一部が D の担当となる．サーバ A

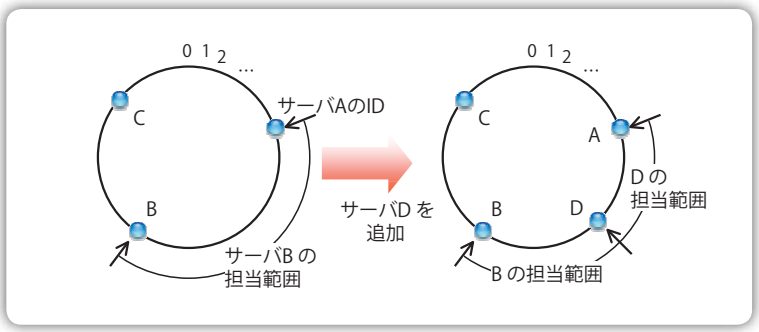


図 -4　サーバが増えた場合の担当範囲の変化（時計回り方式の場合）

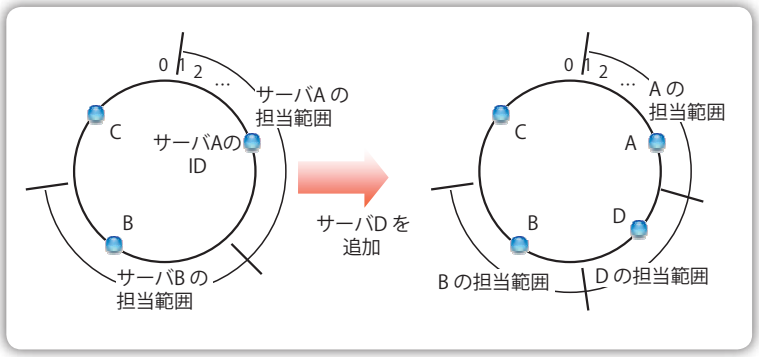


図 -5　サーバが増えた場合の担当範囲の変化（最近値方式の場合）

ら取得したデータなどをキャッシュするデータストアが含まれる．規模としては，数台から多い場合で数百台での動作を狙う．

後者，DHT とは，peer-to-peer，つまり，多数のコンピュータ群が対等な関係で連携して達成するハッシュ表である．正確には，データ構造がハッシュ表である必要はないので，ハッシュ表というよりは key-value ストアである．規模としては数百万台やそれ以上を狙う．

両者は別種のソフトウェアと見なされているが，実のところ，1 つの枠組みの中に位置づけることが可能であり，有効である．以降，具体例を紹介しつつ，両者の関係を論じていく．

■サーバ側データストア

memcached は，ネットワーク越しにアクセスする key-value ストアのサーバソフトウェアである．名前の通り，主に，DBMS などから取得したデータをメモリ上にキャッシュする目的に使われる．Web での著名なサービス，たとえば Facebook，YouTube，Twitter，Digg，はてな，mixi などで使われてきている．mixi では 100 台以上での動作実績があり，Facebook では 1,000 台近くで合計 28 テラバイトのメモリ容量という実績がある．

memcached へはクライアントライブラリを通してアクセスすることが一般的である（**図 -6** (a)）．クライアント，とあるが，これはたいてい，Web サーバ側で動作する Web アプリケーションである．memcached 自体は複数台で連携する機能を持たない．キーに対する担当サーバはライブラリが決める．ライブラリには，ハッシュ値に基づいて決めるライブラリ，コンシステントハッシングで決めるライブラリなど，さまざまなものがある．

Dynamo[3] は Amazon 社が開発した key-value ストアである．同社の Web 上サービスで使われているという．当初の目標は数百台での動作であり，ショッピングカートといったきわめて高い可用性を求められる応用を見据えている．非常に弱い一貫性モデルである結果整合性（eventual consistency）を採用し，不整合の解決は vector clocks を参照して利用側が行う点，また，クライアントとの間で SLA（service level agreement）について合意する点が特徴である．

各キーを担当するサーバはコンシステントハッシングで決まる．読み書きの担当サーバへの誘導は，クライアントライブラリが行う（図 -6 (a)）こともサーバが行うとC に影響はない．

コンシステントハッシングの条件を満たす方法はほかにもある．**図 -5** は，キーの ID から数値的に最も近い ID を持つサーバを担当サーバとする手法を表している．

分散ハッシュ表（DHT）・構造化オーバレイネットワークの方式で言うと，Chord が前者の時計回り方式，Pastry が後者の最近値方式を採用している．コンシステントハッシングの条件を満たす担当サーバ決め手法はほかにもあり，リング状の ID 空間が必須というわけでもない．DHT の方式で言うと，Kademlia，CAN などは他の距離空間を採用している．

サーバ側 vs. peer-to-peer

数台という規模を大きく超えたスケールアウトを狙うデータストアでは，前述の通り，コンシステントハッシングが非常に有用である．コンシステントハッシングを応用したデータストアには，次の 2 種類が見られる．

- サーバ側で動作するデータストア
- peer-to-peer のデータストア：
  分散ハッシュ表（DHT）・構造化オーバレイ

前者にはデータベース管理システム（DBMS）自体や，トランザクション機構を持たない，たとえば DBMS か

(図 -6 (b))こともできる．

Microsoft Azure Platform は，Micorsoft 社が 2008 年 10 月に開発者会議 PDC 2008 で発表したクラウド基盤である．Azure 向けのソフトウェアは，Microsoft 自身が運営するデータセンタだけでなく，従来通り，各組織・個人が自身で運用するサーバ上でも動作させられる．これによってクラウド以前のユーザや開発者の Azure への移行を支援する．

Azure も，データストアにコンシステントハッシングを活用している．それどころか，PDC 2008 での当初発表によれば，次に述べる peer-to-peer 由来の技術である DHT・構造化オーバレイを各所で活用する．たとえば，Blob，Table，Queue といったデータ構造を扱う Windows Azure Storage[☆1] の基礎部分は DHT そのものである．

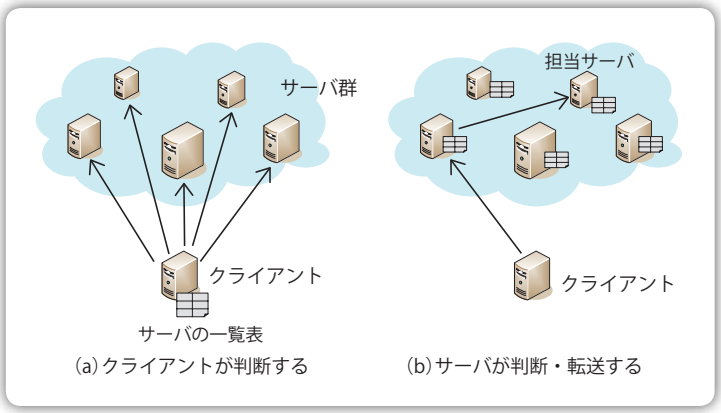


図 -6　サーバ側データストア：担当サーバへの到達方法

■peer-to-peer のデータストア―さらなるスケーラビリティ・耐故障性のために

ここまで説明したサーバ側データストアでは，どこかに全サーバの一覧表があることを前提としている．図 -6 (a) では全クライアントが，(b) では全サーバが全サーバの一覧表を持つ必要がある．(a) に描かれているクライアントは 1 台だが，実際の運用では複数のクライアント(アプリケーションサーバ)が動作する．

サーバの台数が増えるに従って，散在する一覧表を最新の状態に保つことが困難になってくる．サーバを落とす際，サーバ側から一覧表を更新できればよいが，故障などで意図せず落ちる場合を想定すると，表を管理する側からサーバに対しての生存確認は必須となる．実際は存在しないサーバが表に載っていた場合，存在しないことはタイムアウト待ちでしか検知できず，この待ち時間がアクセス性能を大きく損ねるからである．

ここでたとえば，サーバ一覧表を持つコンピュータのうち 2 台が代表してサーバ 100 台の生存確認を行う状況を考える．生存確認のたびに問合せと応答合わせて 400 のメッセージが飛ぶので，5 秒に一度生存確認を行うとすると，毎秒平均 80 のメッセージが飛ぶことになる．80 は許容できたとしても，1,000 台で 800，10,000 台で 8,000 メッセージ／秒を許容できるだろうか．どう工夫しても，秒あたりのメッセージ数は台数に比例せざるを得ない．また，サーバ追加の際は，一貫性を保ちつつすべてのサーバ一覧表を更新しなければならない．完璧を期すためには分散トランザクションが必要となり，一覧表を持つコンピュータの台数に応じてコスト高（可用性が犠牲），そしてほとんど不可能になっていく[☆2]．

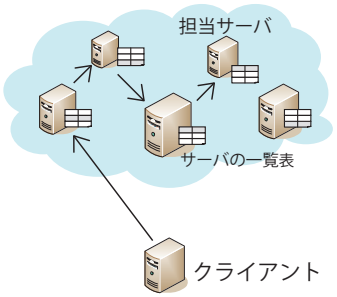
分散したサーバ一覧表すべての間で一貫性を保とうとするから大変になる．ここで，一貫性の条件を緩める方法がある．データストアに対する問合せのサーバ間での転送を許すのである（**図 -7**）．つまり，データの格納，読み出し時に，問合せを受けたサーバが担当サーバではなかった場合，ほかの，より適切と思われるサーバに問合せを回す．サーバ自身が問合せを転送してもよいし，問合せ元に対してより適切なサーバを知らせてもよい．前者の方式を再帰探索（recursive lookup），後者を反復探索 (iterative lookup) と呼ぶ．探索のほかに，フォワーディング，ルーティングと言うこともある．

転送を許すことで，すべてのサーバ一覧表を常に完璧にしておく必要はなくなる．これによって，サーバ一覧表の維持管理がずいぶんと楽になる．メッセージ数を減らせるというだけではない．転送の結果として担当サーバにたどり着ければよいため，全サーバ一覧表の一貫性を保持しなければならない，という制約を緩めることができる．また，一覧表に全サーバを載せる必要もなくなる．表のサイズは，多くの方式で，サーバ数 N として $O(\log N)$ となる．

ただし，転送の結果として担当サーバにたどり着けることは保証されていないと困る．それを保証する方式が peer-to-peer 由来の分散ハッシュ表 (DHT) である．ルーティング，つまり転送の経路決め方式という観点では構造化オーバレイネットワークと呼ばれる．両者の関係は，キーを担当するサーバへのルーティング機能を構造化オーバレイが提供し，それを使って DHT を実装できる，というものである．

転送を繰り返して担当サーバに到達するということは，つまりはルーティングなのである．ここまでの用語をル

☆1　リレーショナルデータベースである SQL Azure とは異なる．
☆2　複製の作り方によっては，完璧な一貫性までは必要ないようにすることもできる．この種のシステムに複製は必須だが本稿では論じない．



(c)図 -6 (b)を元に，サーバ間での転送を許可

図 -7　転送を許す―分散ハッシュ表(DHT)・構造化オーバレイ

	ルーティング用語
サーバ一覧表	経路表
一覧表の維持管理	ルーティング
転送	フォワーディング

表 -1　本稿での用語とルーティング用語の対応

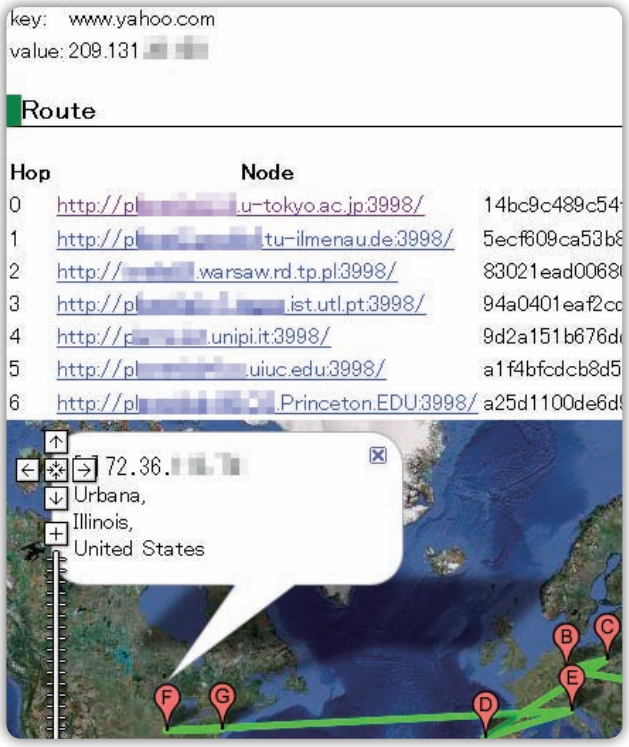


図 -8　PlanetLab 上の 660 台強での
DHT 実装(Overlay Weaver)の動作

ーティングの用語に対応させると，表 -1 のようになる．

構造化オーバレイ・DHT のルーティング方式にもさまざまなものがある．2001 年に CAN，Chord，Pastry，Tapestry が提案されて以来，数多くの方式が提案されてきた．方式によって，経路表の構造や転送先を決めるアルゴリズムが異なる．

応用も進んでおり，たとえば，BitTorrent プロトコルを用いるファイル配信ソフトウェアのうち 20 を超えるものが Kademlia という DHT の一方式を実装している．これによってトラッカというサーバなしでの動作を可能にしている．

筆者らは Overlay Weaver[4] という構造化オーバレイ・DHT の実装を配布しており，これを用いて PlanetLab 上の 660 台強で DNS を模す実験を行っている(図 -8)．

クラウドと構造化オーバレイネットワーク

転送を許すということは，問合せ時の遅延が増えかねないということである（図 -9）．スケーラビリティ，サーバの頻繁な出入りへの耐性といった利点が遅延による損を上回るのはどのくらいの規模からだろうか．採用する方式や実装，応用によるとしか言えず，オーダだけで論じられるものではない．また，Azure の当初発表にも含まれていたように，ルーティングの結果をキャッシュするなど，転送のコストを軽減する手法もさまざまに考えられるため，その効果も考慮に入れる必要がある．

構造化オーバレイという枠組み

全サーバの一覧表を要する「転送なし」と，「転送あり」の DHT・構造化オーバレイは別のものとして扱われてきた．サーバ側データストアの開発者は転送を考慮せず，オーバレイの研究者は「転送なし」を構造化オーバレイには含めずに考えてきた．

しかし「転送なし」は構造化オーバレイの特殊形と位置づけることができる（図 -10）．実際のところ，構造化オーバレイにおいて経路表のサイズを大きくしていくとそれは「転送なし」に近づいていく．たとえば，Chord の successor list を，Pastry の leaf set を大きくしていくといった具合である．このことは，構造化オーバレイという枠組みにおいて「転送なし」と「転送あり」のトレードオフを細かく調整できる可能性，そのような方式を設計できる可能性を示唆している．すると，サーバ側データストアであっても，サーバ一覧表の一貫性条件を緩める方策として，複製だけでなく「転送あり」―ただし経路長は短く抑える―を採る，という設計が合理的になってくるのではないかと筆者は考える．

クラウドを「作る」

国内ではスケールアウトを指向した分散データストアの開発が非常な盛り上がりを見せている．kumofs（古橋貞之氏），ROMA（楽天（株）），Dynamo クローンの

転送なし(図-6)	転送あり(図-7)
• 担当サーバに直接到達	• 担当サーバまで転送数回
• サーバ側 • 低遅延 • 一覧表に全ノードが載る • 全一覧表の一貫性維持 • ～1,000台？	• peer-to-peer 由来 • 小さな経路表O(log N) • 経路表間の一貫性は緩くてよい → サーバの頻繁な出入りに耐える • ～数百万台以上？

図 -9　「転送なし」「転送あり」それぞれの利点

経路表サイズ	経路長	
O(1)	O(log N)	constant-degree DHT
O(log N)	O(log N / log log N)	
O(log N)	O(log N)	一般的なDHT, 構造化オーバレイ
$O(N^{1/2})$	O(1)	
…	…	
O(N)	1	転送なし

図 -10　「転送なし」の構造化オーバレイとしての位置付け

Kai（以上，コンシステントハッシング採用），Flare（グリー（株）），repcached（KLab（株））などを国内の技術者が中心となって開発している．また，2009 年 6 月に開催された Interop クラウドコンピューティングコンペティションでは参加 10 チーム中 5 チームが分散データストアを披露した[5]．開発者のうち最年少は高校 1 年生という若さであった．

とはいえ，そういったスケールアウト指向データストアも，クラウドを構成する要素技術の一部に過ぎない．他の分野も依然おもしろい．

- （サーバやネットワーク等リソースの）ユーティリティ化
  - 仮想化
  - リソース管理
- 分散システム
  - スケールアウト（本稿）
  - …
- …

彼らに続いて，作れる，作ろう，という気持ちを持っていただくことが本稿の隠された目的である．敷居はそう高くない．

参考文献

1）Karger, D. et al. : Consistent Hashing and Random Trees : Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, In Proc. 29th ACM Symposium on Theory of Computing (May 1997).
2）Karger, D. et al. : Web Caching with Consistent Hashing, In Proc. WWW8 (May 1999).
3）DeCandia, G. et al. : Dynamo : Amazon's Highly Available Key-value Store, In Proc. SOSP 2007 (Oct. 2007).
4）Shudo, K. : Overlay Weaver : An Overlay Construction Toolkit, http://overlayweaver.sf.net/
5）中田　敦 : memcached を超える成果も，Interop で若手技術者がクラウドを支える技術を競う，日経 IT Pro, http://bit.ly/ojRrG (June 2009).

（平成 21 年 9 月 23 日受付）

首藤 一幸（正会員）
shudo@is.titech.ac.jp
2001 年早稲田大学大学院理工学研究科情報科学専攻博士後期課程修了．同年産業技術総合研究所研究員．2006 年ウタゴエ（株）取締役最高技術責任者．2008 年より東京工業大学大学院情報理工学研究科数理・計算科学専攻准教授．