〈クラウドの事例紹介〉

2

Googleのクラウド技術

中田秀基　産業技術総合研究所

検索技術とクラウド技術

Google は一般には検索サービスの会社として知られていますが，クラウド提供者としても最大のプレイヤの 1 つで，Gmail や Google Docs，Google Apps 等の SaaS 型クラウド，Google App Engine と呼ばれる PaaS 型クラウドの提供を行っています．

Google がクラウド提供分野にいち早く進出できたのは，検索サービスで培われたスケーラブルな大容量データ蓄積・処理機構がそのままクラウドに転用できたからです．実際，Google のクラウドはサーチエンジンとデータ処理基盤を共有しています．

本稿では，Google の持つスケーラブルな大容量データ蓄積・処理機構を紹介します．まず，Google のデータ蓄積処理機構全体像を概観します．次に，Google の処理基盤を用いたクラウドサービスである Google App Engine（GAE）を紹介します．

Google のデータ蓄積・処理基盤

Google の計算機基盤に関する基本的なコンセプトは，高価で高性能，高信頼のハードウェアを用いる代わりに，安価で低信頼のハードウェアを用いることにあります．性能は量でカバーし，信頼性はソフトウェアで担保します．いくら高信頼のハードウェアを用いたところで，総数がある程度以上になれば，故障は避けられません．たとえば，MTBF（mean time between failure）が 3 年のハードウェアであっても，それが 1000 台あれば，ほぼ毎日どれか 1 つが壊れることになります．

いずれにしろ故障が避けられないのであれば，故障を通常の動作の範囲内としてとらえ，ソフトウェアで解決したほうが賢いと言えます．そのように考えると，価格性能比の良くない高価なハードウェアをわざわざ使う必要はなく，安価なハードウェアを利用したほうがよい，というのは当然の帰結です．このアプローチの利点は，スケールアウト性です．単純に機器の台数を増やすことで容量を増やしていくことができるのです．Google のデータ蓄積・処理基盤はすべてこのコンセプトに従って設計されています．もちろん，このアプローチを実現するためには，非常に高度なソフトウェア技術が必要になりますが，それが現在の Google の競争力の源泉であると言えるでしょう．

Google のデータ基盤を構成するソフトウェアスタックを図 -1 に示します．Google File System と呼ばれる大容量データに特化した分散ファイルシステムがすべての基盤となっており，その上に分散データストアである BigTable が実現されています．並列データ処理基盤である MapReduce もこの Google File System を用いて実装されています．

Google File System

GFS（Google File System）は 2003 年に Google 初の学術論文[1]で発表されました．それ以前から現在に至るまで，10 年近く Google の屋台骨を支えてきたファイルシステムです．

GFS は File System という名前がついてはいますが，通常のファイルシステムに存在する機能が一部サポートされておらず，完全なファイルシステムではありません．たとえばユーザのホームディレクトリを GFS 上に作るようなことは想定されていません．その代わり，大規模なデータを安全に処理することに特化して実装されています．GFS は，クライアント，1 台のマスタ，多数のチャンクサーバから構成されます（図 -2）．クライアントはアプリケーションプログラムです．マスタは，ファイルとファイルを分割したチャンク，チャンクの複製の配置などのメタ情報を管理します．チャンクサーバは，マスタの指令に応じて，チャンクをローカルディスクに保持します．

GFS に書き込まれたファイルは，64M バイトのチャ

ンクに分割され，マスタが決定したチャンクサーバに収められます．この時，各チャンクはデフォルトで合計 3 つの複製が作成され，異なるチャンクサーバに納められます．したがって，あるチャンクサーバが停止しても，ファイルの情報は失われません．

あるチャンクサーバが停止すると，そのチャンクサーバに納められていたチャンクの複製は失われます．マスタは，個々のチャンクに対してそれぞれ 3 つずつ複製を維持しようとしますので，失われていない複製から新たな複製を作成し，別のチャンクサーバに保存します．

このように自動的に複製の個数を保持する機構があるので，GFS に保持したデータは，そう簡単に失われることはありません．もちろん，あるチャンクの複製を納めた 3 つのチャンクサーバが同時に停止してしまえばそのチャンクは失われてしまうわけですが，そのような確率は非常に低いと言ってよいでしょう．

BigTable

BigTable[2] は，GFS の上に作られた大規模データを対象とした分散ストレージで，Google 社内では非常に広く使われています．論文では，アプリケーションとして Google Earth, Google Analytics などの名前が挙げられています．BigTable は一般的なデータベースと異なり，納められたデータに対する検索操作が一切実装されておらず，キーに対する簡単なスキャン操作のみが可能です．このように機能が限定されている代償として，BigTable は非常にスケーラブルに実装されています．

BigTable の基本的なデータ構造は次のようになっています．

（行キー, 列キー, タイムスタンプ）　→　データ

格納されるデータは，単なるバイト列となります．行と列に対してデータを取り出せるという点では一般的なデータベースと変わりませんが，一般のデータベースのテーブルが密（すべての行，列の組合せにデータが入る）であるのに対して，BigTable では疎であることが前提になっています．

BigTable 上のデータは，行キーに対してソートされて保持されます．連続する行キーの範囲に対してスキャンすることができます．

また，ある行キーに属するデータに対する読み書きはアトミック（不可分）であることが保証されます．

BigTable のテーブルは，行キーを単位に分割して保持されます．分割した単位をタブレットと呼び，この単位で GFS への書き込みを管理します．

BigTable の構成図を図 -3 に示します．BigTable は，クライアント，マスタ，タブレットサーバから構成されます．BigTable の構成は，GFS の構成とよく似ていますが，タブレットサーバは GFS のクライアントであり，ローカルディスクに書き込んでいるわけではないことに注意してください．タブレットサーバの物理ノードにタブレットの情報が納められているわけではないので，タブレットサーバとタブレットの対応は固定的ではなく，自由に割り当て直すことが可能です．

もう 1 つの相違は，マスタとクライアントが直接通信せず，Chubby[3] と呼ばれる小容量データとロック機構に特化した特殊なファイルシステムを介して通信していることです．これによって，マスタへの負荷集中を防ぐと同時に耐故障性を得ています．

MapReduce

MapReduce[4] は Google の持つ汎用の並列データ処理機構です．その名の示す通り，MapReduce は，Lisp の

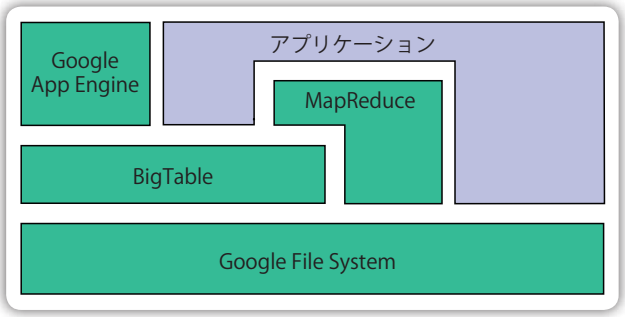

図 -1　Google のデータ蓄積・処理機構

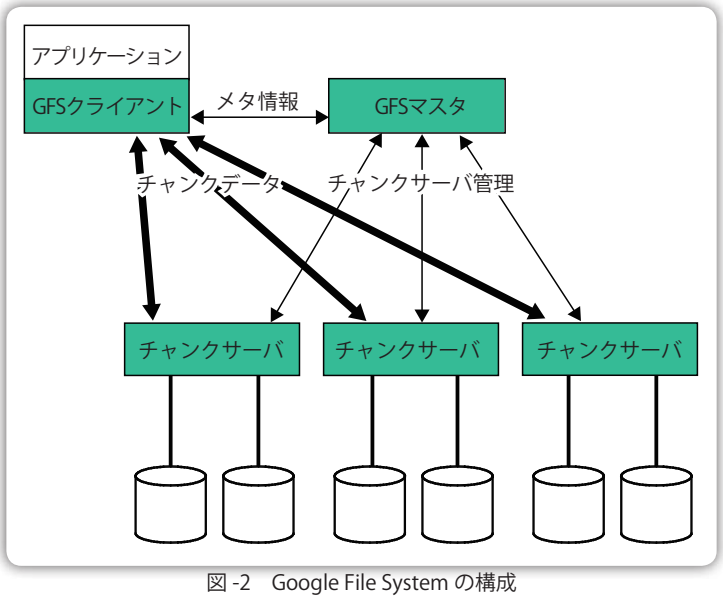

図 -2　Google File System の構成

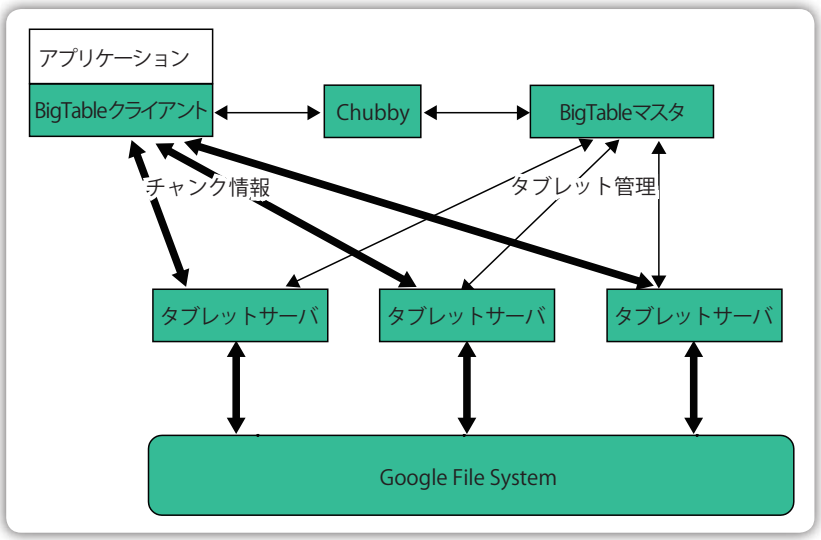

図 -3　BigTable　の構成

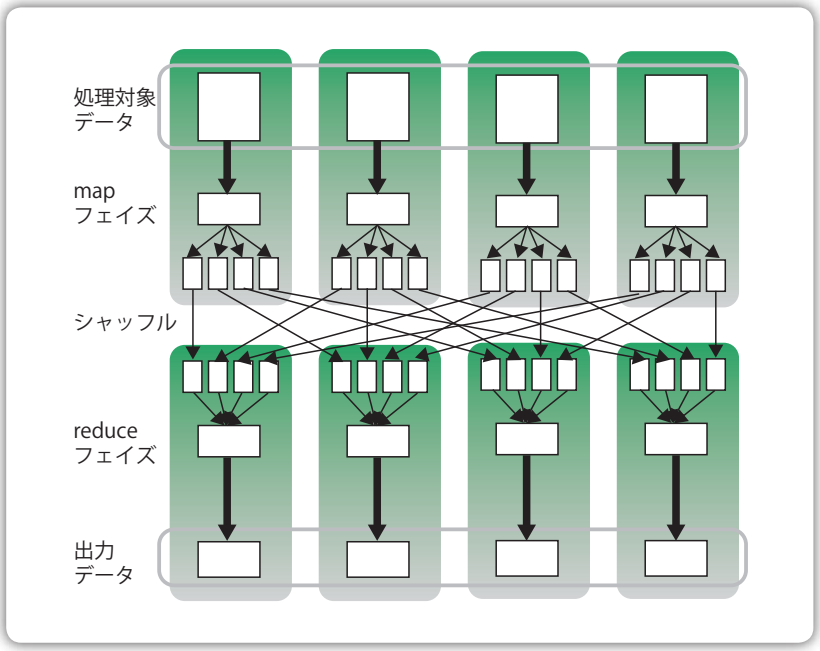

図 -4　MapReduce の概要

map と reduce という高階関数にヒントを得て設計されています．

Lisp の map はリストの各要素に対して同じ操作を行い結果のリストを得る操作で，たとえば，リスト（1 2 3 4 5）に対して，2 倍するという操作を map すると，（2 4 6 8 10）というリストが得られます．reduce はリストの各要素に対して，カスケード的に操作を行い，結果を得ます．たとえば，上記のリストに対して，加算で reduce を行うと，（2+4+6+8+10）=30 が得られます．

MapReduce の map フェイズでは，大量のデータに対して並列に同じ操作を行い，reduce フェイズでは，map フェイズの出力をまとめて最終的な結果を導出します．map フェイズの入力や出力，reduce フェイズの出力は，key と value からなる単純な構造を持ちます．map フェイズの出力は，ソート，シャッフルされてから reduce フェイズにわたされます．reduce フェイズも並列化することが可能です（図 -4）．

たとえば，大量のドキュメント全体に対して，単語の出現頻度を調べることを考えてみましょう．map フェイズでは，各ドキュメントに対して，個別に単語の出現頻度を導出します．各ドキュメントは完全に独立ですから，この操作は並列に実行することができます．次に，各ドキュメントの単語出現頻度をマージして全ドキュメントの単語出現頻度を求めます．これが，reduce フェイズになります．この際，map の結果をあらかじめ単語ごとにマージしておくことで，reduce フェイズも並

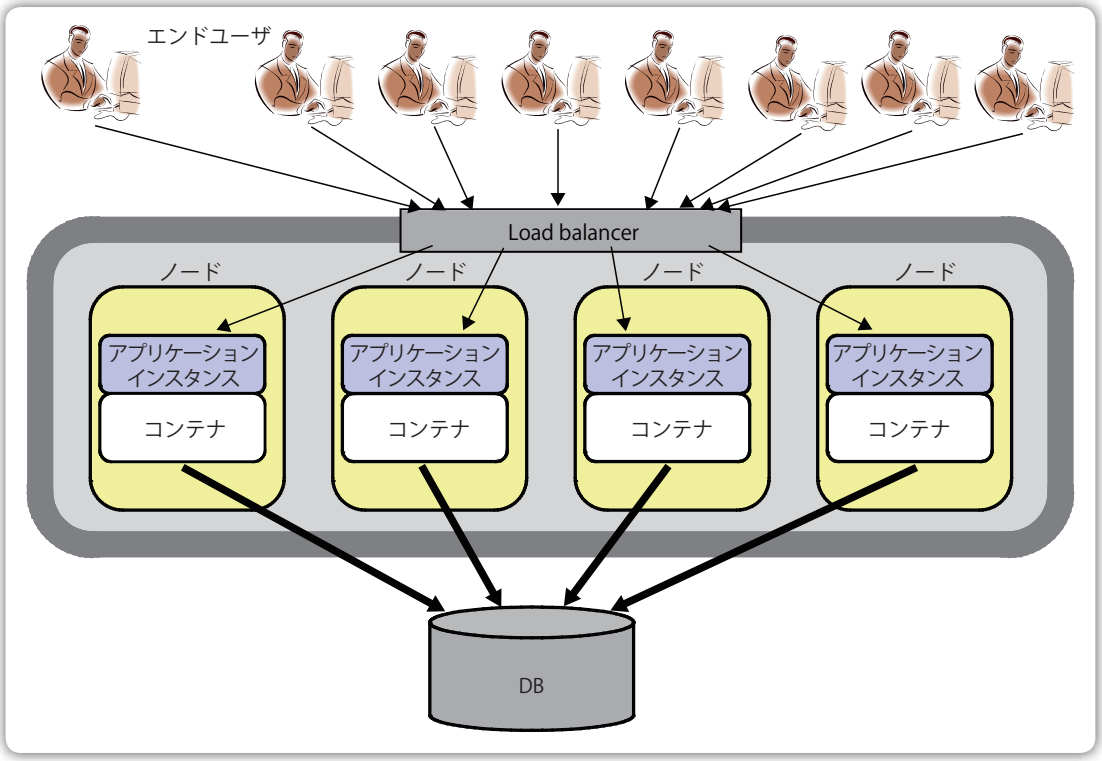

図 -5　WebApplication サーバの構成

列に実行することが可能になります．

MapReduce は GFS から直接データを読み出すように実行することもできますし，BigTable を利用するように実行することもできます．GFS を直接利用する場合には，map フェイズを対象データのチャンクが納められたノードで実行することで，ネットワークアクセスを低減することが可能になります．BigTable を介して利用する場合には，データは同様に GFS においてあるわけですが，かならずタブレットサーバを経由してアクセスすることになるため，性能は低下します．しかし，BigTable 上で行う一連の処理の一部に MapReduce を利用できるというメリットは非常に大きいと思われます．

MapReduce は一般に，C++ のライブラリとして提供され，ユーザは C++ で map，reduce の各関数を記述します．また，Sawzall[5] というある種のスクリプト言語からも利用することができます．Sawzall では，MapReduce の Map フェイズのみがプログラム可能で，Reduce フェイズは，複数用意されている Aggregater の中から選択することになります．したがって，MapReduce の機能を完全に利用することはできないのですが，非常に簡便に利用できるため Google 内部では広く利用されているようです．

Google App Engine と BigTable

Google App Engine は PaaS 型のクラウドサービスで，Web アプリケーションを容易に記述するための枠組みをクラウド上で提供するものです．サービス提供者が，Google App Engine の枠組みに従って，サービスを記述するだけで，スケーラブルなサービスが自動的に構築されます．現在，Python と Java 言語がサポートされており，Python では Django（http://www.djangoproject.com）に類似したフレームワークが，Java では Servlet と JDO（Java Data Object）および JPO（Java Persistent Object）に基づいたフレームワークが提供されています．

Google App Engine で記述されたサービスは，負荷に対して自動的にスケールアウトします．すなわち，複数のサービスコンテナに自動的にデプロイされ，負荷分散されて実行されるのです．通常このような場合に問題になるのは，データベースです．通常のデータベースをバックエンドに用いていると，サービスが複製されても，データベースがボトルネックになり性能が出ません（図 -5）．この問題を回避するために，Google App Engine では，MySQL などのリレーショナルデータベース（RDB）を利用する代わりに，BigTable を利用したデータベースを用いています．これによって，BigTable とその基盤となる GFS のスケーラビリティとアベイラビリティを，すべてのサービス提供者が享受できるのです．

とはいえ，BigTable のデータベースとしての機能は非常に低機能で，リレーショナルデータベースとのギャップが大きいため，直接利用することは困難です．このため，Google App Engine では BigTable に一皮かぶ

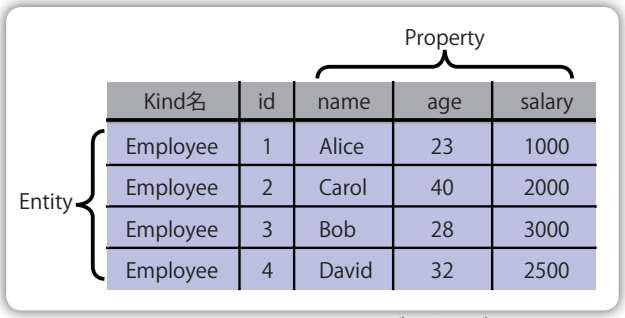

図 -6　Google Application のデータモデル

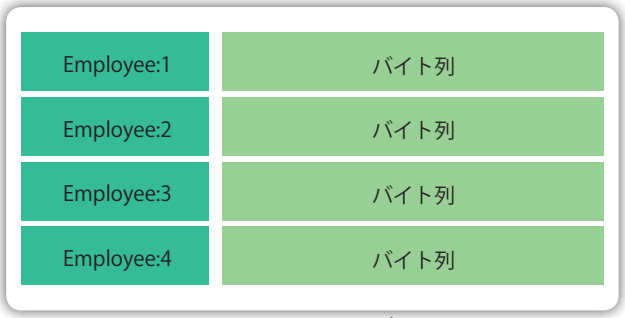

図 -7　Entity テーブル

せ，使いやすくしたものをサービス提供者に提供しています．2008 年の Google I/O でのプレゼンテーション（http://sites.google.com/site/io/under-the-covers-of-the-google-app-engine-datastore）に基づいて，Google App Engine のデータストアの実現方法を紹介します．

まずユーザに提供される，Google App Engine のデータストア構造を見てみましょう．App Engine のデータストアの基本要素は，Entity と呼ばれる構造です．Entity には複数の Property（名，値のペア）を収めることができます．複数の Entity をまとめたものを Kind と呼びます．例として，従業員の名前と年齢と給与額を管理するデータ構造を図 -6 に示します．Kind は RDB のテーブルに，Entity はタプルに，Property は属性に対応すると言うこともできるでしょう．ただし，RDB ではテーブルと属性集合がスキーマで固定されているのに対して，App Engine の Kind と Property の関係は柔軟で，個々の Entity に対して自由に Property を定義することができます．

App Engine では，各 Property に対するレンジ検索が可能です．例に示した従業員であれば，たとえば 25 歳から 35 歳までの従業員のみを列挙することができます．

App Engine のデータストアは，複数の BigTable 上の Table に展開されます．Entity テーブルは，アプリケーションに存在するすべての Entity の情報を保持します．キーは，Entity の Kind と Entity の id を連結したものになります．このとき，Entity の持つ Property 情報は ProtocolBuffer と呼ばれる Google 独自の手法でマーシャリングされ，ただのバイト列として納められます（図 -7）．

これだけでは，Property に対する検索ができません．これを補うために別のテーブル Single-property index テーブルを用います．このテーブルは，すべての Kind，プロパティに対して，Kind とプロパティ名，プロパティ値を並べたものを鍵とし，Entity キーを値とするテーブルです（図 -8）．BigTable 上では，データは鍵に対してソートされて保持されますので，このテーブルのエントリは個々のプロパティに対してソートされていることになります．

たとえば年齢が25歳以上35歳以下の従業員検索は，BigTable 上のキーに対するスキャンになりますので，BigTable の機能を用いて効率的に実装することができます．

Single-property index テーブルは昇順，降順がそれぞれ用意されます．また，この Single-property index テーブルだけでは，複数の Property に関する検索処理を実現することはできません．このため，検索処理に応じたテーブルを個別に生成することで，複雑な検索に対応しています．

たとえば，年齢と給与の双方に対する検索を行うためには，Salary と Age からなる composit index テーブルを用います．これによって，「30 歳で給与が 1,000 万円以上の従業員」を検索することができます．ただし，App Engine のデータストアでは，複数の property に対する不等号による検索はできません．たとえば，「30 歳以下で給与が 1,000 万以上の従業員」は検索できないのです．このような複雑な検索は，データストアの外部でアプリケーションが行わなければなりません．

このように，App Engine のデータストアには多くの制約があり，RDB と比較すると遙かに貧弱であるため，ユーザがその機能を補ってやる必要があります．しかしその代償として，ユーザは GFS と BigTable に由来するスケーラビリティとアベイラビリティを教授できるのです．

GFS の今後

ACM の Web サイト acm queue に掲載された GFS 技術者との対談「GFS:Evolution Fast Forward」（http://queue.acm.org/detail.cfm?id=1594206）によれば，約 10 年にわたって Google の根幹を支えてきた GFS ですが，そろそろ限界が来ているようです．そもそも GFS の最初の利用目的は収集した Web ページデータとインデックス情報の保持でした．これらの処理はバッチ的であるため，GFS はもともとバッチに適したスループッ

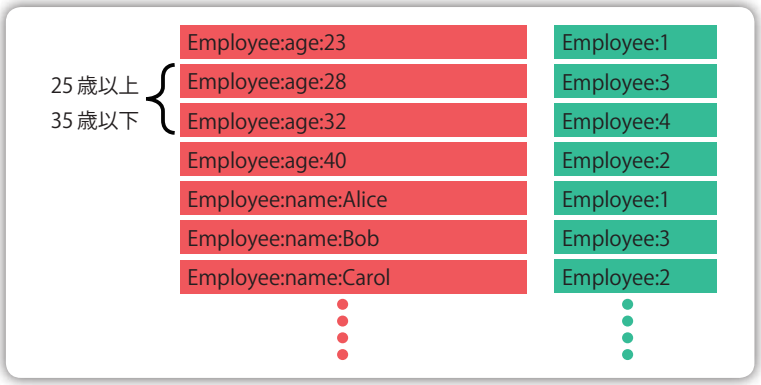

図 -8　Single-property テーブル

ト指向のシステムとして設計されています．ところが，Gmail や Google App Engine などのクラウド型のアプリケーションは，ユーザとのインタラクションを前提とする，レイテンシ指向なアプリケーションです．このため，GFS はこのようなアプリケーションの基盤としてはもともと不適なのです．

性能的な面での問題も顕在化しています．1 つの問題は，マスタがボトルネックになることです．現状の GFS ではマスタは単一です．つまり，すべてのクライアントが 1 つのマスタにアクセスするので，マスタに過大な負荷がかかるのです．もう 1 つの問題はファイル数の制約です．GFS ではすべてのファイルに対するメタデータをマスタノードのメモリ上に保持します．このため，マスタのメモリ容量によって，保持できるファイルの数が決まってしまうのです．GFS は本来，比較的少量の大きいファイルを扱うアプリケーションを想定していたのですが，大量の小さいファイルを扱うアプリケーションが増えてきたため，この問題が顕在化しているようです．

次世代の GFS は，複数のマスタ（数百台）を利用するよう変更されるようです．複数のマスタにメタデータを分散することで，ボトルネックを回避すると同時に，ファイル数の問題を回避します．現時点までに，すでに 2 年開発を続けているそうですので，今後徐々に置き換わっていくのではないでしょうか．

詳しく知るには

本稿では，Google の持つデータ蓄積処理機構を紹介しました．紙面が限られているため各機構の紹介が駆け足になりましたが，詳しく書かれた元論文を参照ください．日本語の資料としては，文献 6）に詳細に紹介されていますので，興味のある方にはぜひご一読をお勧めします．また，Google は Google IO，Google Developer Day と呼ばれる技術コンファレンスを開催していますが，これらでのプレゼンテーション資料やビデオは Web 上で公開されています（http://code.google.com/events/io, http://code.google.com/intl/ja/events/developerday/2009/sessions.html）．参考にしてください．

参考文献

1）Ghemawat, S., Gobioff, H. and Leung, S.,：The Google File System, Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp.20-43 (2003).
2）Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E.：Bigtable：A Distributed Storage System for Structured Data, 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp.205-218 (2006).
3）Burrows, M.：The Chubby Lock Service for Loosely-coupled Distributed Systems, 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)(2006).
4）Dean, J. and Ghemawat, S.：MapReduce：Simplified Data Processing on Large Clusters, OSDI'04：Sixth Symposium on Operating System Design and Implementation, pp.137-150 (2004).
5）Pike, R., Dorward, S., Griesemer, R., and Quinlan, S.：Interpreting the Data：Parallel Analysis with Sawzall, Scientific Programming Journal, Vol.13, pp.277-298 (2005).
6）西田圭介：Google を支える技術―巨大システムの内側の世界，技術評論社 (2008).

（平成 21 年 9 月 1 日受付）

中田秀基（正会員）
hide-nakada@aist.go.jp
産業技術総合研究所情報技術研究部門主任研究員．並列分散計算，グリッド技術，クラウド技術に興味を持つ．博士（工学）．