Vol.2013-OS-126 No.5
2013/7/31

サーバ機能をクライアントに委譲するプライベートクラウドファイルシステム

宮澤 元[1,a)]　青木 勇貴[1]　坂 亮平[1]　筑紫 嵩大[1]　長谷川 浩之[1]　加藤 駿一[1]　加藤 卓[1]

概要：本稿では、ストレージ資源をプライベートクラウドに集約する環境において、クライアントの計算資源を活用するファイルシステムについて述べる。本システムでは、ファイルシステムのサーバ機能のうちメタデータ管理と複製管理の一部をクライアント計算機に委譲することができる。サーバ機能の一部をクライアントに委譲することにより、クライアントの計算資源の有効活用を図るとともに、サーバ負荷の低減が期待できる。

A Private Cloud File System that Delegates Server Functions to Client Hosts

Hajime Miyazawa[1,a)]　Yuki Aoki[1]　Ryohei Ban[1]　Takahiro Chikushi[1]　Hiroyuki Hasegawa[1]
Shunichi Kato[1]　Suguru Kato[1]

Abstract: This paper describes a file system that utilizes computing resources of client hosts in private cloud environment where storage resources of an organization are consolidated. The system allows users to delegate a part of server loads of metadata and replication managements to client hosts. It is expected that the delegation not only effectively utilizes unused computing resources of client hosts but also lowers server loads.

1. はじめに

計算資源を集約するためにクラウドを利用することが一般的になっている。インターネットを介して利用するパブリッククラウドだけではなく、組織内の計算資源を集約するプライベートクラウドの利用も進んでいる。

従来、組織内の計算機利用環境は図 1 に示すような構成をとることが普通であった。すなわち、組織全体で共用されるサーバ群に加え、組織の各部門ごとに利用される部内サーバが存在し、これらをクライアントとなる計算機が利用する環境である。このような環境では、組織全体として共用サーバを管理するのに加え、各部門ごとに部内サーバを管理する必要があり、管理コストが高い。

一方、プライベートクラウドを導入すると、共用サーバや部内サーバをクラウドに集約することによって、計算資源の管理コストの低減が期待できる。従来の共用サーバや部内サーバの機能はクラウド内で仮想的に提供されるので、クライアントは従来と同等の計算機環境を利用することができる (図 2)。

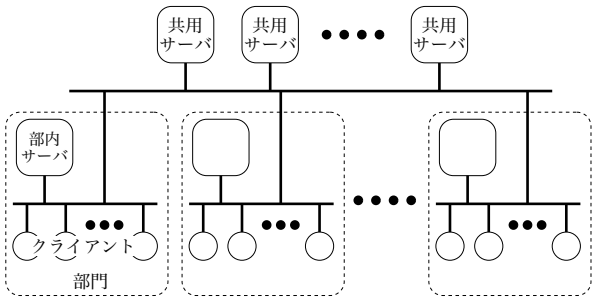

図 1　従来の組織内計算機環境

しかし、計算資源の集約という観点から見ると不十分な点がある。クライアントとして利用される計算機はサーバ

[1] 南山大学 情報理工学部 ソフトウェア工学科
Department of Software Engineering, Faculty of Information Sciences and Engineering, Nanzan University
[a)] miyazawa@nanzan-u.ac.jp

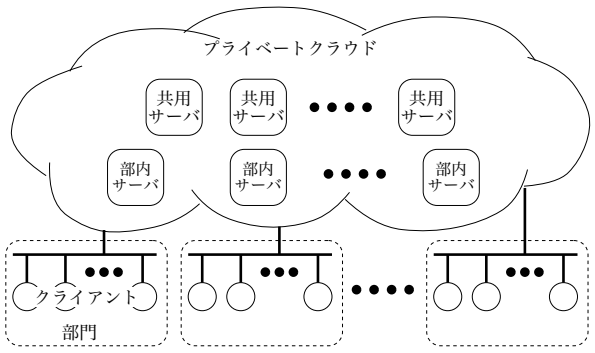

図 2 プライベートクラウドを用いた組織内計算機環境

と違って常に高負荷で運用されることは稀であり、計算資源に余裕があると考えられるが、プライベートクラウドを利用する場合でも、クライアントは利用者側に残ったままで、計算資源として集約することはできない。組織内で利用されるという性格上、クライアントも組織の管理下にあることが普通である[*1]が、組織全体としてみた時、クライアントの計算資源が積極的に活用されているとは言えないことになる。

本稿では特にストレージ資源をプライベートクラウドに集約する環境でクライアントの計算資源を活用するファイルシステムについて述べる。ファイルシステムのサーバ機能のうち、メタデータ管理とストレージ管理の一部をクライアント計算機に委譲することにより、クライアントの計算資源の有効活用を図るとともにサーバ負荷を低減できる。

2. プライベートクラウドファイルシステム

クラウドで利用されることを想定したファイルシステムは、Google File System[1] や Hadoop HDFS[2] をはじめとして多数提案されている。これらのファイルシステムは、主にパブリッククラウドで大規模データの処理に利用されることを想定している。

一方、我々が想定するようなプライベートクラウドでは、このような大規模データ処理を行うケースもある程度はあるものの、部内サーバが提供する NFS サーバのようなファイルサーバを集約することが求められるようなケースが多いと考えられる。

そこで我々は、通常の UNIX ファイルシステムインタフェースを備え、プライベートクラウド環境で動作するファイルシステムを試作することとした。本システムの設計にあたっては、Ceph[3], [4], [5] を参考にした。

3. 本システムの概要

本システムは、プライベートクラウド側のサーバホストで動作するメタデータサーバとストレージサーバ、および各クライアントホスト上で動作するクライアントエージェントからなる (図 3)。

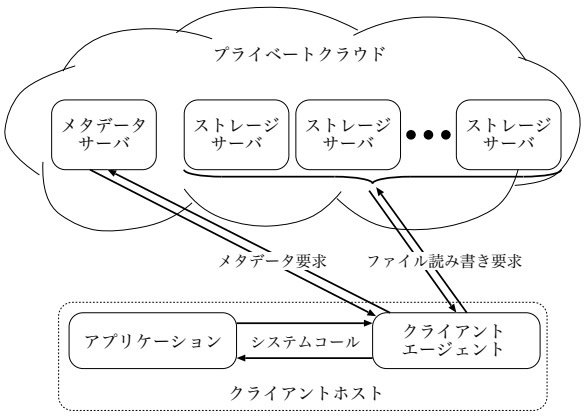

図 3 本システムの構成

3.1 メタデータサーバ

メタデータサーバは、ファイルのメタデータ管理を行うとともに、名前空間の管理を行う。クライアントエージェントからの要求に応じて inode 番号をはじめとするファイルのメタデータを返す他、ファイルの新規作成や削除、更新などに対応したメタデータ管理を行う。メタデータをストレージサーバ上のファイルとして格納し、永続性を確保する。メタデータサーバは、メタデータ管理を委譲されたクライアントエージェント (4.1 節参照) を除けば、システム全体に 1 台のみ存在するので単一障害点となる。これを防ぐため、将来的には何らかの多重化機構を導入することを検討中である。

3.2 ストレージサーバ

ストレージサーバはファイルデータの管理を行う。プライベートクラウド上でストレージを提供する全てのサーバホストで動作する。各ストレージサーバは、それぞれがクライアントエージェントからのファイル読み書き要求に応じることができ、自ホストが提供するストレージに対してファイルデータの読み書きを行う。

本システムでは、ファイルは 128MB 単位のチャンクから構成されている。ストレージサーバは各チャンクをローカルファイルシステム上のファイルとしてストレージに格納する。

ファイルの可用性と安全性を向上するために、ストレージサーバはファイルが更新されると、プライマリコピーレプリケーション [6] 風のアルゴリズムを用いて他のストレージサーバと通信し、複数のサーバホスト上にファイルの複製を作成する[*2]。

[*1] 最近では BYOD(Bring Your Own Device) が注目されるなど、必ずしもこのように言い切れるものではない。

[*2] 標準では 3 台の計算機上に複製を保持する。

3.3 クライアントエージェント

クライアントエージェントは、本システムが提供するファイルへのアクセスを行うアプリケーションが動作するクライアントホストで動作する。アプリケーションからのファイルシステムに対する要求を受け取り、必要に応じてメタデータサーバ、ストレージサーバと通信して要求を処理する。

現在の実装では、クライアントエージェントは FUSE[7] を用いてマウントされたファイルシステムを経由して、アプリケーションが発行したシステムコールによるファイルアクセス要求を受け取ることができる。

3.4 ファイルアクセス要求の処理

本システムにおけるファイルアクセス要求の処理の流れを示す。

アプリケーションは UNIX の通常のファイルアクセス手順に従ってファイルアクセスを行う。つまり、ファイルを open() してから read(), write() を用いて読み書きを行い、ファイルを close() する。

クライアントエージェントは、アプリケーションのファイル open() に応じて、メタデータサーバにファイルメタデータを要求する。メタデータサーバは、自身が管理するメタデータを検索して要求されたファイルが存在するかどうかを調べ、存在すればそのメタデータを返す。

クライアントエージェントは、マッピング関数を用いてメタデータサーバから返されたメタデータに含まれる inode 番号とファイルのチャンク番号から、該当のチャンクを保持する全てのストレージサーバのリストを得ることができる。通常、クライアントエージェントはリストの先頭のストレージサーバをプライマリとして利用する。

アプリケーションの read(), write() 要求に対して、クライアントエージェントはプライマリストレージサーバに該当のチャンクを要求する。プライマリストレージサーバは、ローカルファイルシステム上のチャンクを読み出し、クライアントエージェントに返す。クライアントエージェントは、ローカルファイルシステム上にチャンクのキャッシュを作成し、実際のファイル読み書きはこのキャッシュに対して行う。

アプリケーションからファイル close() 要求を受け取ると、クライアントエージェントはメタデータサーバにメタデータの更新要求を出す。読み出しアクセスしか行われなかった場合でも、ファイルのアクセス時刻などが更新されるのでメタデータを更新する必要がある。

ファイル close() 時にファイルデータに書き込みがあり、キャッシュが更新されている場合には、ストレージサーバにも書き戻しを行う。クライアントエージェントは、プライマリストレージサーバに更新されたキャッシュを書き戻す。プライマリストレージサーバは更新をローカルディスクに書き込むとともに、指定された台数のセカンダリストレージサーバにもファイルの更新を要求する。全てのセカンダリストレージサーバから更新完了の通知を受け取ると、プライマリストレージサーバはクライアントエージェントに更新完了を通知する (図 4)。

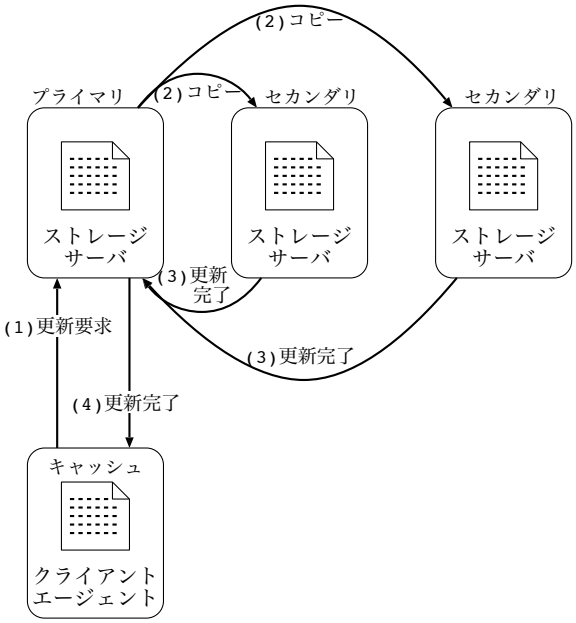

図 4 ストレージサーバにおける複製管理

同一ファイルに書き込みを行う複数のクライアントエージェントが存在する場合、ファイル一貫性の問題が生じる可能性がある。本システムでは Andrew File System(AFS)[8] における callback と同様の一貫性制御を行い、session semantics 相当の一貫性を保持している。具体的には、ファイルデータが更新されたファイルが close() された場合、そのファイルのキャッシュを持つ他のクライアントエージェントに対して、メタデータサーバがキャッシュの無効化を要求し、キャッシュを削除させる。

4. サーバ機能の委譲

本システムでは、クライアントの計算資源を活用するために、メタデータサーバにおけるメタデータ管理とストレージサーバにおける複製管理の機能の一部をクライアントエージェントに委譲できる。

4.1 メタデータ管理の委譲

ファイルシステムの特定のサブツリーを指定して、そのサブツリーに含まれるファイルのメタデータ管理を特定のクライアントホスト上で動作するクライアントエージェントに委譲する。委譲されたサブツリーに含まれるファイルのメタデータはメタデータサーバではなく委譲されたクライアントエージェントが管理する (図 5)。メタデータ管理

を委譲されたクライアントエージェントは、該当するファイルに対するメタデータ要求を、メタデータサーバに要求することなく自分で処理することができる。

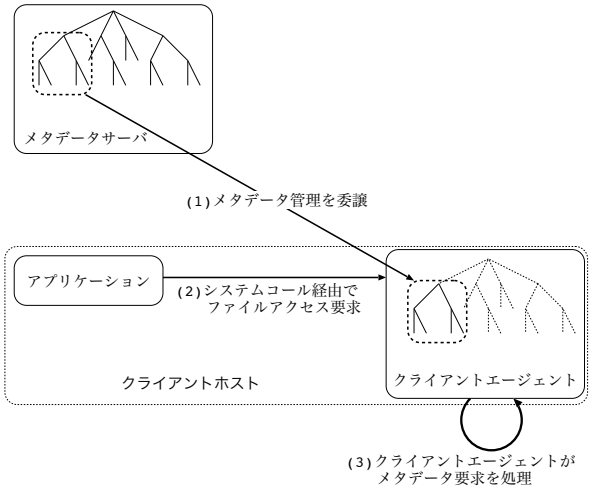

図 5 メタデータ管理の委譲

4.1.1 メタデータの委譲処理

メタデータ管理の委譲は、クライアントエージェントからの要求に従って行われる。委譲要求を受け取ると、メタデータサーバは指定されたサブツリーとクライアントホストの組を委譲テーブルに記録し、その後、このサブツリーについてのメタデータ要求は受け付けない。メタデータ自体はストレージサーバ上のファイルとして格納されているので、メタデータ管理を委譲されたクライアントエージェントは、必要なメタデータをストレージサーバから読み取り、メタデータ要求に応えることができる。

現在の実装では、メタデータのアクセスやメタデータ管理の委譲について、特別なアクセス制御は行なっていないが、将来的には適切なアクセス制御機構を導入すべきである。

4.1.2 メタデータ要求のリダイレクション

あるサブツリーについてのメタデータ管理が委譲されていることは、メタデータサーバと移譲されたクライアントエージェントだけが知っているので、他のクライアントエージェントが委譲されたファイルについてのメタデータをメタデータサーバに要求することがありうる。このときメタデータサーバは、要求元のクライアントエージェントに対して委譲先のクライアントエージェントを通知する。そこで、要求元のクライアントエージェントはメタデータ要求を委譲先のクライアントエージェントに出し直す。委譲先のクライアントエージェントはこの要求に答えてメタデータを要求元のクライアントエージェントに返す (図 6)。

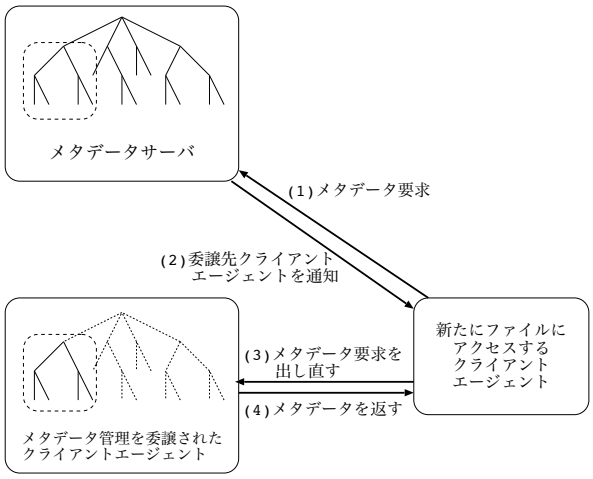

図 6 メタデータ要求のリダイレクション

4.2 複製管理の委譲

クライアントエージェントは、ファイルシステムの特定のサブツリーを指定して、そのサブツリーに含まれるファイルの複製管理をストレージサーバに代わって行うことができる。3.4 節で述べたように、ストレージサーバはプライマリコピーレプリケーション風のアルゴリズムでファイルの複製管理を行なっており、プライマリストレージサーバからセカンダリストレージサーバへのファイルデータの送信や、セカンダリストレージサーバからの更新完了通知待ちを行なっている (図 4)。クライアントエージェントがファイルデータをストレージサーバに書き込む際、プライマリコピーレプリケーションを行わないようにオプション指定を行い、代わりにクライアントエージェント自身が必要な全てのストレージサーバにファイルデータを書き込むように動作することができる (図 7)。これを複製管理の委譲と呼んでいる。

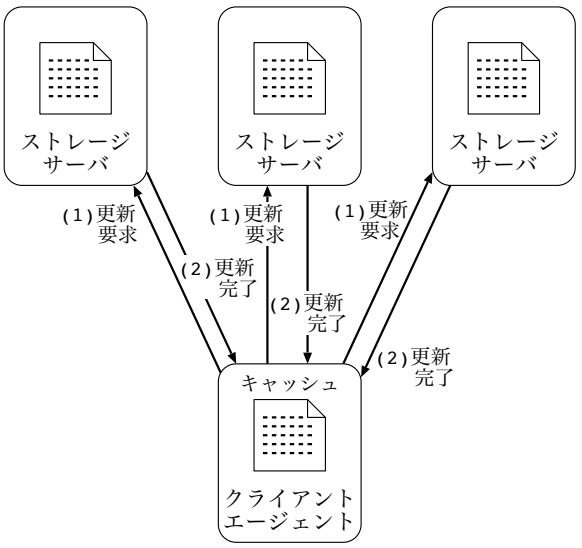

図 7 複製管理の委譲

メタデータ管理の委譲とは異なり、プライマリストレージサーバはクライアントエージェントが最初にアクセスするというだけで、何か特別な情報を管理しているわけではない。あるファイルデータの書き込み先のストレージサーバリストは、inode 番号とチャンク番号からマッピング関数を用いて得ることができるので、どのクライアントエージェントでも知ることができる。従って、複製管理の委譲を行っても何かの権限がクライアントエージェントに委ねられるわけではなく、あるファイルについての複製管理の委譲を複数のクライアントエージェントに対して行うことも可能である。また、複製管理を委譲されたクライアントエージェントとそうでないものが混在していても特に問題は生じない。

4.3 障害対策

クライアントホストはサーバホストと比較して耐故障性が低いと考えられる。また、クライアントホストは故障していなくても利用していないときには停止させておくことが多く、サーバ機能の委譲を行うにあたって、障害対策を行う必要がある。

4.3.1 障害発生時の対応

メタデータ管理の委譲については、障害が問題となるのはメタデータ管理を委譲されたクライアントエージェントの停止中に、他のクライアントエージェントがメタデータ要求を送信した場合である。このメタデータ要求はエラーとなるので、メタデータ要求を出したクライアントエージェントは、メタデータサーバに委譲先のクライアントエージェントが停止中である旨のフラグとともに再度メタデータを要求する。この要求を受け、メタデータサーバは委譲先クライアントエージェントへのメタデータ管理の委譲を中止する。メタデータそのものは、ストレージサーバ上に格納されているので、委譲を中止したサブツリーについてもメタデータサーバが引き続きメタデータ管理を行うことができる (図 8)。

複製管理の委譲については、クライアントエージェントに特別な権限が委ねられるわけではない。このクライアントホストが障害などで停止してもシステム全体の動作に影響をおよぼすことはないので、特別な障害対策を必要としない。

4.3.2 障害復旧時の対応

障害などで停止していたクライアントホストが復旧する場合、メタデータ管理の委譲に関して、停止前の状態を保持している可能性がある。クライアントエージェントは、起動時にこの状態が正当なものであるかどうかをメタデータサーバに問い合わせる。停止中に委譲が中止されておらず、保持している状態が正当なものであることが確認できれば、クライアントエージェントはメタデータ管理を委譲された状態のまま動作を再開できる。停止中に委譲が中止されていれば、クライアントエージェントが保持している停止前の状態は破棄され、必要であれば再度メタデータ管理の委譲を受ける。

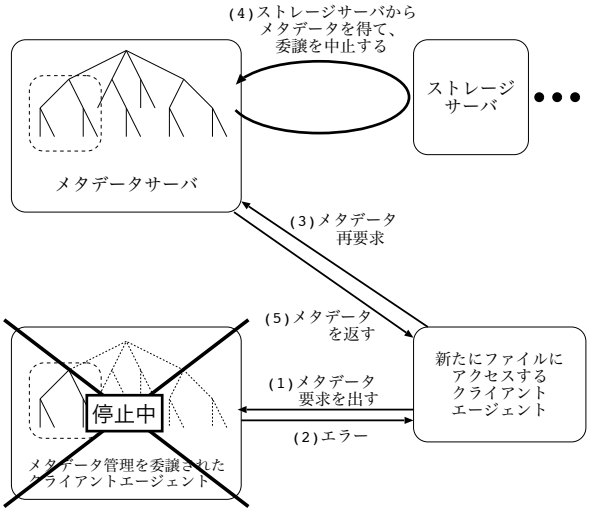

図 8 メタデータ管理を委譲する際の障害対策

複製の委譲については復旧時に特別な処理を行う必要はないが、クライアントが保持しているファイルデータのキャッシュに関しては正当性の確認を行う必要がある。

5. おわりに

本稿では、ストレージ資源をプライベートクラウドに集約する環境でクライアントの計算資源を活用するファイルシステムについて述べた。ファイルシステムのサーバ機能のうち、メタデータサーバにおけるメタデータ管理とストレージサーバにおける複製管理の一部をクライアント計算機に委譲することにより、クライアントの計算資源の有効活用およびサーバ負荷の低減を図る。

現在、本システムのプロトタイプの実装を Ubuntu Linux 12.10(カーネルバージョン 3.5.0, 64bit) が動作する PC 上で行なっており、メタデータサーバおよびストレージサーバの主な機能は動作している。今後、クライアントエージェントの実装を進め、本システムを用いた場合のサーバおよびクライアントの負荷状況の変化などについて実験を行う予定である。

参考文献

[1] Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP'03)*, pp. 29–43 (2003).

[2] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST'10)*, pp. 1–10 (2010).

[3] Weil, S. A., Brandt, S. A., Miller, E. L. and Long, D. D. E.: Ceph: A Scalable, High-Performance Distributed File System, OSDI'06: 7th USENIX Symposium on Op-

erating Systems Design and Implementation, pp. 307–320 (2006).

[4] Weil, S. A., Brandt, S. A., Miller, E. L. and Maltzahn, C.: CRUSH: controlled, scalable, decentralized placement of replicated data, *Proceedings of the 2006 ACM/IEEE conference on Supercomputing(SC '06)*, New York, NY, USA, ACM, (online), DOI: 10.1145/1188455.1188582 (2006).

[5] Weil, S. A., Leung, A. W., Brandt, S. A. and Maltzahn, C.: RADOS: a scalable, reliable storage service for petabyte-scale storage clusters, *Proceedings of the 2nd international workshop on Petascale data storage(PDSW '07)*, New York, NY, USA, ACM, pp. 35–44 (online), DOI: 10.1145/1374596.1374606 (2007).

[6] Alsberg, P. A. and Day, J. D.: A principle for resilient sharing of distributed resources, *Proceedings of the 2nd international conference on Software engineering (ICSE '76)*, Los Alamitos, CA, USA, IEEE Computer Society Press, pp. 562–570 (online), available from ⟨http://dl.acm.org/citation.cfm?id=800253.807732⟩ (1976).

[7] Szeredi, M.: File System in User Space (2006). http://fuse.sourceforge.net/.

[8] Spasojevic, M. and Satyanarayanan, M.: An Empirical Study of a Wide-Area Distributed File System, *ACM Transactions on Computer Systems*, Vol. 14, No. 2, pp. 200–222 (1996).