Vol.2016-IOT-35 No.11
Vol.2016-SPT-20 No.11
2016/9/24

インタークラウドを用いた高可用性安否確認システムの基礎評価

永田正樹[1,2]　阿部祐輔[2]　福井美彩都[2]　磯部千裕[2]
長谷川孝博[3]　峰野博史[1]

概要：災害時における市民の安否情報の収集および公開を目的とする安否確認システムは確実な稼働を要求される．昨今の安否確認システムは災害時の停止を回避するためオンプレミスで構築するのではなく，クラウドを用いた実装が主流である．筆者らは AWS(Amazon Web Services)を用い，システムを構成する各サーバを複数地点に分散配置し可用性向上を実現した広域冗長型安否システムを提案し一定の成果を得た．通常，分散システムを構築する場合、各サーバの連携を考慮し単一クラウドベンダーでの実装が一般的である。しかし単一クラウドベンダーでの運用では，当該ベンダーの都合によるシステムメンテナンスや障害発生等でシステム全停止の懸念がある．本論文では，各サーバを複数クラウドベンダー間で構築し，データベースに分散データ管理が可能な Cassandra を用いインタークラウド構成での実装および分散稼働を評価した．インタークラウドを用いることで単一障害点をなくし高可用性を実現した．

キーワード：安否確認システム，インタークラウド，高可用性，分散データ管理

An Evaluation of High Availability Safety Confirmation System Using Intercloud

MASAKI NAGATA[1,2]　YUSUKE ABE[2]　MISATO FUKUI[2]　CHIHIRO ISOBE[2]
TAKAHIRO HASEGAWA[3]　HIROSHI MINENO[1]

Abstract: Safety confirmation system for the purpose of collecting and publishing of citizens safety information at the disaster is required to ensure operation. Recently developed safety confirmation systems are based on cloud computing is a mainstream rather than on-premises in order to avoid system stop at the disaster. The authors have obtained certain results by the proposed a global redundant web safety confirmation system that realizes the availability improved distributed each server to multiple regions using the AWS(Amazon Web Services). In case of build a distributed system, implementation of a single cloud vendor is common in consideration of the cooperation of each server. However, a single cloud vendor there is a risk of the entire system stops at the failure caused or system maintenance by the vendor convenience. In this paper, we evaluate the implementation and the distributed operation of intercloud configuration to build each of the servers across multiple cloud vendors using the Cassandra. To achieve a high availability eliminates the single point of failure by using intercloud.

Keywords: safety confirmation system, intercloud, high availability, distributed data management

1. はじめに

2011 年の東日本大震災や 2016 年の熊本地震のような多大な被害をもたらした災害時において，効率的な被災者安否の把握がその後の救助および復興速度向上に寄与する．災害時の安否情報の収集および確認の仕組みとして安否確認システム（以下，安否システム）がある．東京都では「東京都帰宅困難者対策条例（平成 25 年 4 月 1 日施行）[1]」において，企業間および家族間での連絡手段確保の事前準備を求めている．また経済産業省での「事業継続計画策定ガイドライン[2]」では，安否確認実施手順の制定化を求めており，これら施策から安否確認の重要性がうかがえる．

安否システムはシステムに利用登録しているユーザ間の安否情報の収集および公開を目的とする．ユーザは PC やスマートフォン等から安否情報の登録や閲覧等のアクセスをおこなうため，昨今の安否システムは Web システムを用いた実装が一般的である[3]．また，安否システムは災害時の持続稼働を要求されるため，システム稼働基盤はユーザ資産のオンプレミスではなく耐災害施策がなされているクラウドベンダーへのアウトソースが主流である．しかしクラウドベンダーへの安易な移管は課題がある．

1 つ目の課題は単一クラウドベンダーでの運用である．昨今のクラウドベンダーは強固な耐災害施策により自然災害には強い耐性を持つ．しかし事業撤退など経営面でのシステム停止はユーザの立場からは防ぐ手段がない．2 つ目の課題はシステムデータの分散管理である．安否システムは無停止稼働を求められるため，災害やその他影響でサーバが停止した際，別サーバでの代替運用を可能とする分断特性に強い機構が必要である．本研究では上記 2 つの課題を解決するため，複数クラウドベンダーを用いたインタークラウド構成と複数サーバでの分散データ管理を可能とした安否システムを試作開発し，その有効性を示す．

1 静岡大学 創造科学技術大学院
Graduate School of Science and Technology, Shizuoka University
2 株式会社アバンセシステム
AvanceSystem Corporation
3 静岡大学 情報基盤センター
Center for Information Infrastructure, Shizuoka University

Vol.2016-IOT-35 No.11
Vol.2016-SPT-20 No.11
2016/9/24

2. 従来研究と課題

2.1 安否確認システム

一般的な安否システムの動作フローは**図 1** となる．まず気象情報提供サービスから災害発生を安否システムが取得する．つぎに安否システムからユーザへ安否報告を促すメールを送信する．つぎにユーザは安否システムに対して安否報告をする．さいごにユーザ間で安否情報を共有する．

文献[4]は，従来はオンプレミス環境で運用していた安否システムをクラウドへリプレースする提案である．クラウドへのリプレースは，オンプレミス環境での災害時の持続稼働の懸念に対して，システム稼働基盤をクラウド環境とすることで可用性の向上を実現している．安否システムを構成する各サーバを分散配置する研究[5]では，ロバストネス向上を目的とし複数サーバを用いたミラーリングでの冗長化や負荷分散の提案がある．このように災害時の持続稼働を要求される安否システムは，可用性向上を目的としたクラウド環境下での運用が必須といえる．

2.2 インタークラウド

本研究でのインタークラウドとは，単一のクラウドベンダーではなく複数のベンダー間でのシステム連携を指す．文献[6]では，複数のベンダー間を連携するサブシステムを構築し，そのサブシステム上にユーザシステムを稼働させるものである．ユーザにベンダー間の差を意識させることなくサービス提供を可能としている．文献[7]では，商用のクラウドベンダーと研究室内に構築したプライベートクラウド環境との連携を試みている．状況に応じたクラウド選択ポリシーを用いてサーバの広域分散配置を実現しシームレスなクラウド環境構築を可能としている．このような先行研究では，いずれも単一クラウドではなく複数クラウドを用いることで可用性向上や利便性向上を実現しており，クラウド間連携の重要性を説いている．

2.3 静岡大学での取り組み

東海大地震の危険地域に位置する静岡大学では，2009 年 5 月から全学に安否システムを導入し，約 7 年の運用を経ている[8]．主な仕様は，システムを構成するサーバのクラウド上での運用，認証コード付き URL を用いたアカウント ID や PW 入力を省略した迅速なログイン認証，教育機関ならではの複雑な組織構成を吸収した無制限組織階層機能，学生・教職員の属性データを管理する学務・教務データベースとの自動連携機能などがある．

静岡大学を含む一般的な安否システムのデータ管理はリレーショナルデータベース（以下，RDB）を用いている．理由はユーザの安否情報や所属学部などの管理に対して，ユーザ ID などをキーとした各属性情報の CRUD 操作（Create, Read, Update, Delete）が容易なためである．しかしシステムのアクセスログをみると，氏名や学年，所属学部などの属性情報への変更や更新アクセスは通常ほぼない．

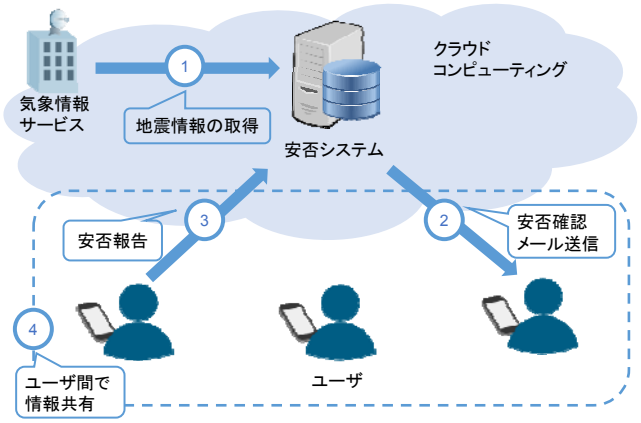

図 1　安否確認システムの動作フロー
Figure 1　Flow of the safety confirmation system.

多頻度でアクセスされる情報は災害時の安否報告である．つまり災害時にはユーザの安否情報を更新するため Update 操作にて相当数の安否報告アクセスが，マスター／スレーブ構成のうちマスター側 DB に集中する．RDB には Update アクセスを分散するハッシュ関数を用いたシャーディング技術があるが，データ規模拡大にともない ID 採番に変更を加える場合やデータ検索の複雑化など，必ずしも利点ばかりではない．また，一般的に RDB は CAP 定理のうち分断耐性（Partition-tolerance）に弱く，分散システムに向かない特性がある．

2.4 課題

これまでを整理すると，安否システムを停止せず持続稼働を実現するための課題としては，複数ベンダーを用いたインタークラウド構成と，複数サーバにデータを分散配置するデータ管理があげられる．

安否システムに限らず Web システムは複数のサーバで構成することが多い．それぞれのサーバを冗長化し可用性を向上する実装は従来から行われてきた．しかし多くは単一ベンダー内での冗長化および可用性向上対策であり，通常災害では効果を発揮しても大規模災害や災害以外での対策としては万全とは言い難い．その理由は，単一ベンダーのみの実装では当該ベンダーの経営面や他都合でのサービス停止の可能性を無視できないことに起因する．つまりシステム停止を回避するためには単一ベンダーではなく複数ベンダーを連携したインタークラウド構成が必要となる．

インタークラウド構成を用いることは必然的に分散システムでの実装となる．RDB は分断耐性に弱く分散システムには不向きであるため，複数サーバが連携する分散データ管理に適しているデータベースシステムが望まれる．分散データ管理では，システムを構成する各ノード間でのデータ同期および障害等であるノードが停止した際でも持続稼働が可能なことが必須である．また災害時は多数の安否報告アクセスがなされるため，分散データ管理だけでなく Update（書き込み）特性に優れた機構が理想的である．

Vol.2016-IOT-35 No.11
Vol.2016-SPT-20 No.11
2016/9/24

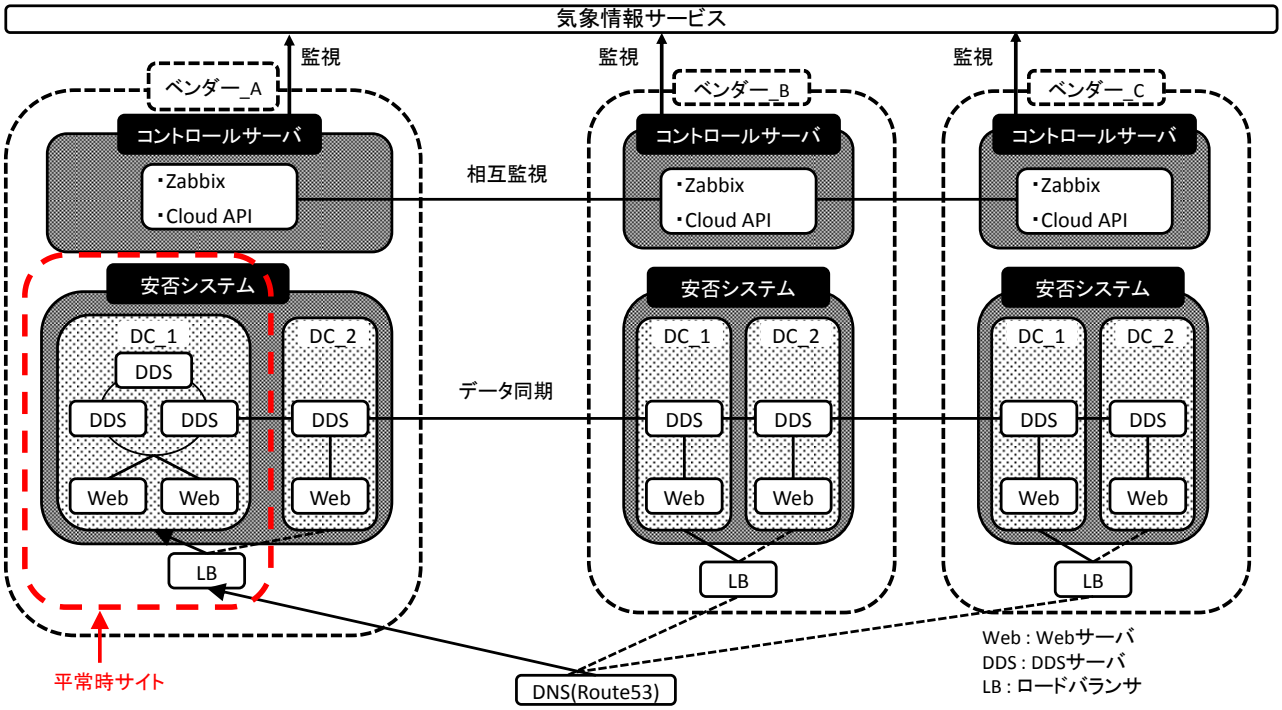

図 2　提案システム

Figure 2　Proposed system.

3. 提案システム

3.1 システム概要

提案システムは，システム基盤に 3 社のクラウドベンダーを用い，**図 2** に示すように各ベンダー間で連携し運用する．複数ベンダーを用いることでベンダー間をまたぐインタークラウド構成となり広域冗長化構成を実現する．各ベンダーの役割としてプライマリベンダーとセカンダリベンダーがある．プライマリベンダーとはシステム正常稼働中のアクセスをすべて引き受けるベンダーのことで，ベンダーA となる．セカンダリベンダーとはプライマリベンダーが何らかの事情で停止した際のバックアップベンダーとなり，ベンダーB，C となる．プライマリベンダーの障害発生時は，セカンダリベンダーにアクセス先を変更しシステム持続稼働を実現する．各ベンダー内にはそれぞれ安否システムとコントロールサーバを配置する．安否システムはユーザの安否情報の収集および公開をおこない，コントロールサーバはベンダー内およびベンダー間でのシステム持続稼働のための諸作業をおこなう．

安否システムは Web と分散データベース（DDB）で構成する．本研究での DDB とは複数サーバを連携してデータの分散管理を実現する仕組みである．ベンダー内の 2 つのデータセンタにそれぞれ Web，DDB を置き，単一ベンダーにおいても閉域冗長構成で運用する．ベンダーA の場合，DC_1 は Web 2 台 DDB 3 台，DC_2 は Web 1 台 DDB 1 台となり，DC_1 がプライマリデータセンタ，DC_2 がセカンダリデータセンタとなる．システム正常稼働時にはすべてのアクセスはプライマリデータセンタに向き，セカンダリデータセンタはプライマリデータセンタ停止時のバックアップサイトである．つまり平時の正常稼働中は図 1 のベンダーA の DC_1 がサービス提供サイトとなる．データセンタ間およびベンダー間でのデータ同期は DDB のレプリケーション機能を用いておこなう．

コントロールサーバは，安否システムやロードバランサなどのクラウドリソースの障害検知および障害時のリカバリを担当する．障害検知は統合監視ソフトウェアの Zabbix[9]を用いておこなう．各ベンダーのコントロールサーバは自ベンダーだけでなく他ベンダーのシステム監視を相互におこない，障害発生時のリカバリを確実に実行することで単一障害点をなくし可用性向上に寄与する．また，安否システムが安否報告収集メールを送信する契機となる災害情報を気象情報サービスなど[10]から検知し，安否システムに通知する．

3.2 インタークラウドでのベンダー間連携

提案システムは，ベンダーA に AWS[11]，ベンダーB に Azure[12]，ベンダーC に Cloudn[13]を用いる．安否システムやコントロールサーバに用いる各サーバは各ベンダーの IaaS である．ベンダー選定の要件として，API での IaaS 起動／停止やロードバランサなどのクラウドリソースのコントロールが可能なことがあげられる．提案システムでは各ベンダーが相互に稼働監視およびリカバリを行うことでシステム停止を回避する.このためシステム稼働監視やリカバリを実行する API を具備し，さらに自動化することがベンダー選定の必須条件となる．本研究で用いた各ベンダー

Vol.2016-IOT-35 No.11
Vol.2016-SPT-20 No.11
2016/9/24

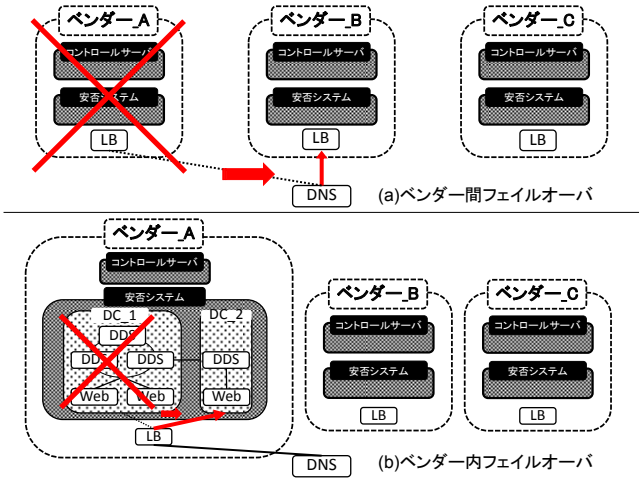

図 3　ディザスタリカバリ
Figure 3　Disaster recovery.

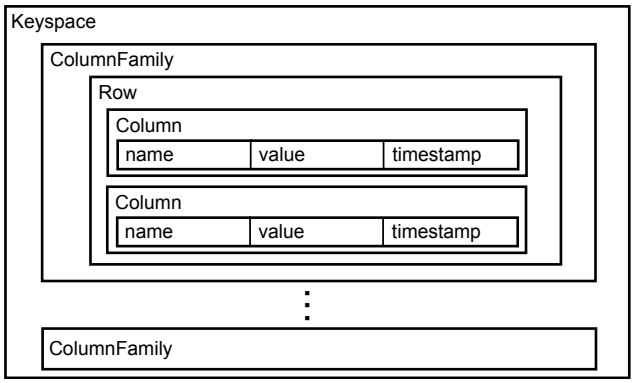

図 4　Cassandra のデータ構造
Figure 4　Data structure of Cassandra.

はこれら条件にあてはまり，自ベンダーだけでなく他ベンダーにおいても Cloud API 経由（図 2）でコントロール可能である．たとえばベンダーB の Azure からベンダーA の AWS にアクセスする場合，ベンダーB のコントロールサーバに AWS API をインストールすることで，ベンダーB 上からベンダーA のクラウドリソースをコントロールする．

システム停止の回避は，複数ベンダーを連携するだけでは実現できず，システム中の単一障害点を排除する実装が必要である．たとえば 3 社のベンダーを用いてシステムを構築したとき，3 社の Web サーバのフロントに 1 社のロードバランサを配置したのでは，このロードバランサが停止した場合はシステム全体が停止する．つまりひとつのベンダーに依存する実装を避け，あるベンダーが停止した際にはシステム稼働を持続するためのディザスタリカバリが機能しなければならない．ディザスタリカバリは一般的に二つの考え方があり，ひとつは障害発生時においてもシステムを停止せずに稼働し続けることを可能とするものと，システムを一旦停止し再稼働をおこなうものがある．安否システムは災害時にこそ稼働を求められ，また本研究ではインタークラウドを用いた無停止システムを目的とするため，前者のアプローチでの実装が必要である．

提案システムには 2 種類のディザスタリカバリ機構がある．ひとつはベンダー間でのディザスタリカバリであり，プライマリベンダーが停止した場合，DNS の設定を変更しセカンダリデータセンタにアクセス先を変更するベンダー間でのフェイルオーバである（**図 3a**）．もうひとつはベンダー内のディザスタリカバリであり，ベンダー内のプライマリデータセンタが停止した場合，ロードバランサの設定を変更しセカンダリデータセンタにアクセス先を変更するベンダー内でのフェイルオーバである（**図 3b**）．両者いずれにおいても無停止でのディザスタリカバリを実現するため各ベンダーおよびベンダー内のサーバ群はホットスタンバイで運用し常にデータ同期をおこなう．この 2 つのディザスタリカバリ機構でベンダー内およびベンダー間それぞれの障害対応を可能とし，システム停止を回避する．

3.3 Cassandra を用いた分散データ管理

従来の安否システムはデータ管理に RDB を用いているため，極力そのデータ構造を流用可能なデータ管理が望ましい．そこで提案システムでは分散データ管理システムのうち，スキーマ定義を必要とする Cassandra[14]を用いる．Cassandra はデータアクセスに SQL 文を用いない NoSQL と呼ばれる分散データ管理システムであり，書き込み特性に優れている[15]．データ管理構造はデータ（Value）に対して一意の標識（Key）にて管理を行う Key-Value Store（KVS）方式であり，Cassandra は複数ある KVS 方式の NoSQL のうち，カラム指向型に属する．カラム指向型とは単純な KVS が Key と Value を 1 対 1 の関係で管理するのに対し，Column と呼ぶ Key と Value の組を Row にて複数管理を可能とし単純な KVS 方式を高度化したものである．Cassandra のデータ構造は**図 4** となる．各データ単位を RDB と対比すると，Keyspace はデータベース，ColumFamily はテーブル，Row はレコードに相当する．この構造は安否システムにおける RDB でのユーザ管理属性スキーマとの親和性が高く，他の NoSQL と比較してデータ管理の移植がしやすい．

提案システムは，災害時の持続稼働や単一ベンダー障害に依存しない稼働を実現するため，Cassandra での分散データ管理が必須となる．Casandra は単体稼働も可能だが，通常は複数サーバを用いクラスタ構成での運用が一般的である．クラスタを構成する各ノードが連携し，あるノードが停止した際でも別ノードがその役割を果たし持続稼働を可能とする．Cassandra のデータ管理は RDB のようなマスター／スレーブの概念を持たず各ノードが等価である．このため単一ノードに依存しないデータ管理を可能とし単一障害点がない．このようなアーキテクチャは，RDB においても複数サーバを用いた読み取り専用のリードレプリカや書き込み分散のシャーディングで可能であるが，これら機能

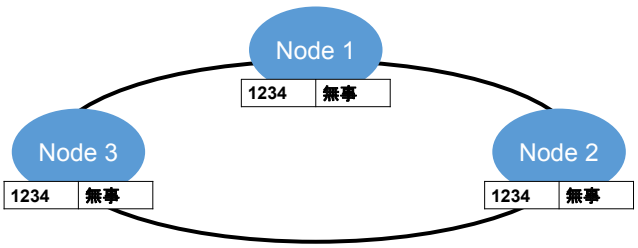

図 5　ノードと replication factor
Figure 5　Node and replication factor.

は多くの RDB では標準機能でなく，ユーザ自身の実装や外部ソフトウェア依存となり管理が複雑となる．一方，Cassandra はクラスタ構成での稼働を前提として設計されているため，外部ソフトウェア等を用いずに RDB におけるリードレプリカやシャーディング相当の機能を実現可能であり，管理面の簡略化に寄与する．

提案システムは**図 5** に示すようにベンダーA の 3 つのノードに対して，replication factor（RF）を 3 で運用する．RF とはデータをコピーする合計数であり，RF が 3 の場合は 3 つのノードにデータのコピーを保持する．図 5 では Key にユーザ ID，Value に安否情報を設定したデータが 3 つのノードすべてにコピーされる．提案システムの場合は，ノードが 3 台で RF も 3 であるため，すべてのノードにデータをコピーすることで可用性を向上する．また，Cassandra のマルチデータセンター機能でベンダーA のノードとベンダーB，C のノードが自動でレプリケーションしディザスタリカバリに備える．このように Cassandra はノード台数，RF 等をシステムに適した設定で運用することで可用性を向上することができ，安否システムのような持続稼働を求められるシステムのデータ管理および保持に適している．

4. 実装と評価

4.1 実装

提案システムで用いるクラウドベンダーとインスタンスタイプは**表 1** となる．**表 2** は各ベンダーのクラウド API であり，各ベンダーのコントロールサーバにそれぞれインストールし自ベンダーだけでなく他ベンダーのコントロールもおこなう．**表 3** は各サーバのシステム環境である．

ディザスタリカバリ実施時の障害検知は各ベンダーのコントロールサーバ上で稼働する Zabbix がおこなう．ベンダー間の障害検知は，各ベンダーの Zabbix が各ベンダー上のロードバランサ経由で安否システムへアクセスし障害を検知する．ベンダーA の安否システムの障害時は，ベンダーB または C のコントロールサーバが aws-cli を用いて AWS Route53 の DNS レコードにアクセス先をベンダーB または C に変更してフェイルオーバを実施する．ベンダー内の障害検知およびフェイルオーバも同要領であり，ベンダーA の DC_1 の障害時は，ベンダーB または C のコントロールサーバ上から aws-cli を用いて「elb register-instances」コマンドにて AWS ELB のアクセス先を変更する．

表 1　クラウドベンダーとインスタンスタイプ
Table 1　Cloud vendors and instance types.

ベンダーA	AWS	t2.small
ベンダーB	Azure	Standard A1
ベンダーC	Cloudn	プラン v1

表 2　クラウド API
Table 2　Cloud API.

AWS	aws-cli 1.10.56
Azure	Azure cli 0.10.3
Cloudn	Cloudn SDK for Ruby 0.0.1

表 3　システム環境
Table 3　System environment.

安否:Web サーバ	CentOS 6.5 Apache 2.2.15
安否:DB サーバ	CentOS 6.5 Cassandra 2.0.6
コントロールサーバ	CentOS 6.5 Zabbix 2.4.7

安否システムの Web は同内容のアプリケーションソースコードおよび OS を各クラウドベンダーの機能であらかじめ OS イメージ化しておき，障害時は表 2 の API を用いて当該ベンダーのロードバランサ配下に起動する．DB も Web と同要領で Cassandra のインストールおよび初期設定済みの OS を各ベンダー内に OS イメージ化しておく．このとき安否関連のデータは OS イメージには含まない．Cassandra は起動時に自身が参加するクラスタに接続しデータ同期をおこなうため OS イメージの時点では安否関連情報は不要となる．Cassandra 起動後，データ同期を経て準備が整うとステータスが「UN」となり運用可能となる．

監視フローは，(1) 安否システム監視用のダミーユーザを作成し定期的に安否報告などを実施するスクリプトを cron 等に設定しアクセス成功可否を Zabbix に返す，(2) 監視結果がエラーの場合は障害サーバ検知のために各サーバへ疎通確認し該当サーバを特定する，(3) 代替サーバを起動する，という流れとなる．たとえばベンダーA の Web サーバが停止した際は，ベンダーB または C のコントロールサーバ上から aws-cli を用いて「run-instances」コマンドを発行し AWS EC2 を起動する．このとき停止した Web サーバと同様のロードバランサ配下に起動する．同様にベンダーB の Web，DB サーバが停止した際は，ベンダーA または C のコントロールサーバ上から Azure-cli を用いて「azure vm start」で Azure Virtual Machines を起動する．

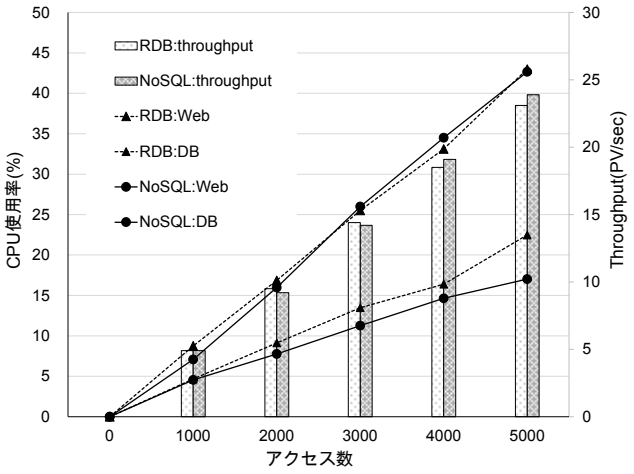

図 6　RDB と NoSQL の性能比較
Figure 6　Performance comparison of RDB and NoSQL.

4.2 評価

インタークラウドでのベンダー間連携の評価では，ベンダー間およびベンダー内でのディザスタリカバリ後の正常稼働を確認した．ベンダー間ではベンダーA のロードバランサを停止し，ベンダーB または C のコントロールサーバが Route53 の設定を変更しベンダーB へのフェイルオーバを確認した．ベンダーB へのアクセス変更は Route53 の DNS が反映する約 70 秒後となり，わずかではあるがシステムが停止する結果となった．ベンダー内ではベンダーA の DC_1 の Web サーバを停止し，ベンダーB または C のコントロールサーバがロードバランサの設定を変更し DC_2 へのフェイルオーバを確認した．この変更はシステム停止時間がなく瞬時に反映可能であった．

Cassandra での分散データ管理では，ベンダーA の DC_1 のノードを 1 台停止した状態で安否情報データが正しく取得できることを確認した．また，ノード停止後，新規ノードを追加し安否情報データが正しく取得できることを確認した．提案システムではノード追加および準備が完了するまでは約 90 秒程度の時間を要した．RDB との性能比較は **図 6** となる．評価環境はベンダーA の DC_1 にてロードバランサ配下に 12 台の Web サーバを配置し，RDB と Cassandra はともに 1 台ずつとし，JMeter[16]を用いて複数の安否報告アクセスを提案システムに対して実施した．RDB は PostgreSQL8.4.12 を用いた. 安否報告アクセスは 10 分間で 0 から 5000 まで増減し，Web サーバ，DB サーバの CPU 使用率とスループットを計測した. CPU 使用率は各サーバ上の sar コマンド，スループットは JMeter での計測である．アクセス数を増加していくと同程度のスループット値でわずかに Cassandra の CPU 使用率が低いが，ほぼ性能は両者とも差がない．しかし Cassandra は通常単体では運用しないため，処理性能面では RDB と同程度でも複数ノードを用いた可用性面に RDB と比較し優位性がある．

5. まとめ

本研究では災害時の安否情報を収集および公開する安否システムの稼働基盤に，複数クラウドベンダーを連携したインタークラウドアーキテクチャを採用し基礎評価をおこなった．各ベンダーの API をそれぞれのベンダーで相互に使用可能とすることで，ベンダー間での連携および冗長構成を実現した. Cassandra を用いた分散データ管理では各ノードにそれぞれデータコピーを保持することで単一ノード停止でのシステム全体停止を回避し可用性向上を実現した.

今後の課題として, Cassandra ノード数での性能向上があげられる．Cassandra はシステムの処理能力が飽和した際，ノードを追加してデータ容量拡大や性能向上が可能なことが知られている．今後はノード数での性能比較を調査する予定である．また，Cassandra 以外の分散データベースシステムの安否システム上での稼働評価をおこなう予定である.

参考文献

1) 東京都防災ホームページ，<http://www.bousai.metro.tokyo.jp/kitaku_portal/1000050/1000536.html>（2016-08-20）.
2) 経済産業省 事業継続計画策定ガイドライン，<http://www.meti.go.jp/report/downloadfiles/g50331d06j.pdf>（2016-08-20）.
3) 梶田将司, 太田芳博, 若松進, 林能成, 間瀬健二: 高等教育機関のための安否確認システムの段階的構築と運用, 情報処理学会論文誌, Vol.49, No.3, pp.1131-1143(2008).
4) Y. Hiroaki, and N. Suzuki: “Development of Cloud Based Safety Confirmation System for Great Disaster”, International Conference on. IEEE, *WAINA 26th*, pp.1069 - 1074(2012).
5) H. Echigo, H. Yuze, T. Hoshikawa, K. Takahata, N. Sawano, and Y. Shibata: “Robust and Large Scale Distributed Disaster Information System over Internet and Japan Gigabit Network”, International Conference on. IEEE, *AINA 21st*, pp.762 – 768(2007).
6) 波戸邦夫, 上水流由香, 岡本隆史, 横山大作,: 複数の異種クラウド間におけるスケールアウトおよびディザスタリカバリ機構の実装とその評価, 情報処理学会デジタルプラクティス, Vol.4，No.4(2013).
7) 神屋郁子, 川津祐基, 下川俊彦, 吉田紀彦: 複数のクラウドを利用したサーバ広域分散配置システムの構築, 電子情報通信学会論文誌 B, Vol.J96-B, No.10, pp.1164 - 1175(2013).
8) 長谷川孝博, 井上春樹, 八巻直一: 低コスト運用でユーザフレンドリな安否情報システムの開発, 学術情報処理研究誌, No.13, pp.91 - 98(2009).
9) Zabbix, <http://www.zabbix.com/> (2016-08-22).
10) 一般財団法人気象業務支援センター，<http://www.jmbsc.or.jp>（2016-08-22）.
11) Amazon Web Services, <https://aws.amazon.com/> (2016-08-22).
12) Microsoft Azure, <https://azure.microsoft.com/> (2016-08-22).
13) Clondn, <http://www.ntt.com/business/services/cloud/iaas/cloudn.html> (2016-08-22).
14) Apache Cassandra, <http://cassandra.apache.org/> (2016-08-22).
15) 松浦伸彦, 大畑真生, 太田賢, 稲村浩, 水野忠則, 峰野博史: 大規模センサデータを処理可能な分散型データ管理システムの提案, 情報処理学会論文誌, Vol.54, No.2, pp.721-729(2013).
16) Apache JMeter, <http://jmeter.apache.orf/> (2016-08-22).