VA Linux Systems 株式会社

CEPH/RADOS

障害発生時のノウハウ

Ceph の現状や課題から、OpenStack 環境で Ceph に障害が発生した場合のシステム環境への影響に関するテスト結果とその対処法について、考察を交えて解説します。

2017 年 8 月執筆、10 月 Web サイト掲載

目 次

0. はじめに

Ceph は OpenStack 環境向け分散ストレージのデファクトスタンダードとして広く利用されています。ブロック IF、オブジェクト IF、ファイル IF を備えており、Ceph のみで OpenStack の様々なコンポーネントが使用するストレージバックエンドを構築できることだけでなく、長期間に渡って安定して動作することや、ネット上に流通している情報が豊富なことで、安心して利用できる点が選ばれているものと思われます。

OpenStack 環境のストレージを Ceph に集約できることは大きなメリットですが、その結果 Ceph の障害が OpenStack 環境に及ぼしうる影響範囲は広くなります。
Ceph 自体の品質が安定していることも大きなメリットですが、その結果システムの運用体制が、Ceph 障害は発生しないことが前提になっている、という声もあるようです。
よって、障害発生時のノウハウは是非とも蓄積・整理しておきたいところです。

本稿では、Ceph を構成する各種ハードウェアで障害が発生してから復旧するまでの間の、OpenStack 環境で走行中のインスタンス (VM) 上でアプリケーション I/O や、システム管理者が OpenStack 環境に対して行ったオペレーション (インスタンスの作成等) への影響を調べます。
これには、発生したハードウェア障害が何であったかを事後に調べられるかという観点や、ハードウェア障害復旧後に OpenStack 環境を復旧するための追加的な手順が必要だったかどうかという観点等も含まれます。

一方、ソフトウェア障害に対しては、現象の状況やログ等の情報を参考に既知の問題かどうかをネット上で検索するところから始まり、未知の問題の場合はコアとソースコードから解析することになります。
Ceph は RADOS と呼ばれる分散オブジェクトストレージと、それを利用する各種ストレージ IF 毎のサービスという構成となっており、使用する機能範囲に応じて調査対象のコード規模は大きくなります。
テストコードや内部的に使用している Ceph 以外のコンポーネント (キーバリューストアやウエブサーバー等) を除いたコード全体の規模は Linux カーネルに含まれるすべてのファイルシステムのコード (linux-4.7/fs) の総量におよそ匹敵します。
よって、ソフトウェア障害発生時に調査すべきソースコードの範囲をできるだけ限定できる仕組みが欲しいところです。

本稿では、Ceph の一連のログから、そのときに動作していた可能性のある処理ルートの候補をリストアップする仕組みの試行についても紹介します。

本稿の取り組みは、2015 年から 2017 年に掛けて、弊社 (VA Linux Systems Japan 株式会社) の VA Quest サービスをご利用のお客様に対して実施したものです。

弊社および VA Quest サービスについては下記 URL を参照ください。
VA Linux Systems Japan 株式会社 http://www.valinux.co.jp/
障害解析・サポートサービス (VA Quest) http://www.valinux.co.jp/services/support/

本稿で使用している Ceph および OpenStack のバージョン (2017 年 8 月末 本稿執筆時点) は以下のとおりです。
2015 年 Ceph Hammer + OpenStack Kilo
2016-2017 年 Ceph Infernalis + OpenStack Liberty
上記より新しいバージョンを使用した場合、本事とは結果が異なる可能性があります。

1. OpenStack/Ceph の現状と課題

1.1. Ceph を選択する利点

OpenStack Summit の開催時期 (半年毎) に合わせて OpenStack コミュニティで選択されている各種技術のアンケート (*1) の結果が発表されます。
この中の、OpenStack ブロック・ストレージ・サービス (Cinder) で使用されているストレージバックエンドのシェアに注目します。
Ceph (rbd) は 2014 年 11 月に首位となって以降も毎回シェアを伸ばしており、2017 年 4 月の結果では全体シェアで 65%、大規模システム (1000 コア以上) で 44%と、実質的にデファクトスタンダードとなっているようです。

Ceph が選択される理由はケースバイケースだと思いますが、以下のような理由が含まれるのではないかと考えます。

- ブロック IF、オブジェクト IF、ファイル IF を備えており、Ceph のみで OpenStack の様々なコンポーネントが使用するストレージバックエンドを構築できる
    - ブロック IF (Nova/QEMU,Cinder, Glance, …)
    - オブジェクト IF (Swift, …)
    - ファイル IF (Manila)
- 安心して利用できる
    - 長期間に渡って安定して動作する
    - ネット上に流通している情報が豊富
- その他 (利用する OpenStack ディストリビューションに Ceph が含まれている等)

(*1) https://www.openstack.org/user-survey

1.2. Ceph を選択する課題

Ceph を選択する利点は課題をともなうと考えられます。
OpenStack 環境のストレージを Ceph に集約できることからくる支出の小ささは大きなメリットですが、その結果 Ceph で発生した障害が OpenStack 環境に及ぼす影響は複合的に広範囲になる可能性があります。
Ceph 自体の品質が安定していることからくる運用費の安さも大きなメリットですが、その結果システムの運用体制が、Ceph で障害が発生しないことが前提となっており、今後何かあった場合に対するノウハウが蓄積されていないという不安の声もあるようです。

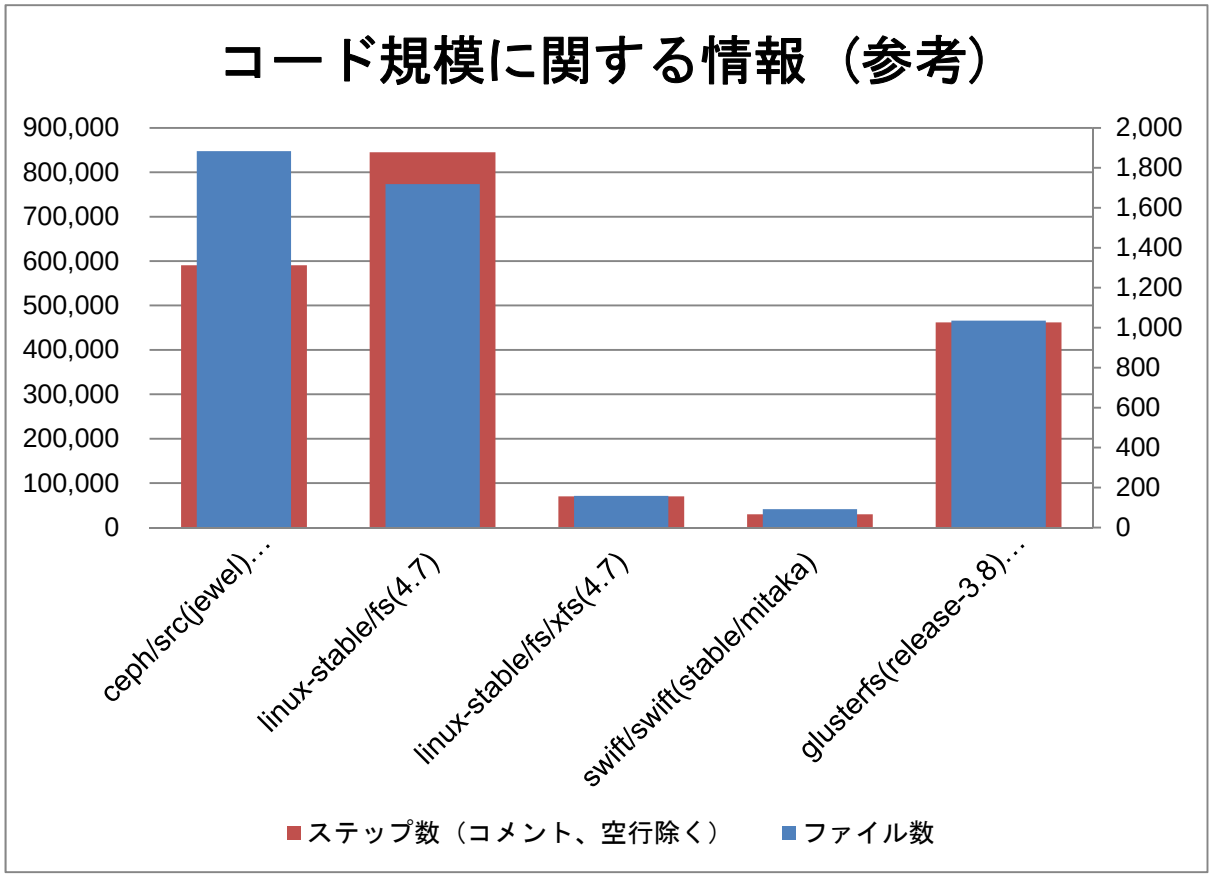
Ceph 障害にはハードウェア障害とソフトウェア障害（バグ）がありますが、上記はその両方に当てはまると考えられます。
ハードウェア障害に対しては、現象から原因を切り分けて故障箇所を特定し、良品と交換することになります。

本稿で紹介する検証で分かったことでもありますが、比較的日常的なハードウェア障害でも対応にはノウハウがあります。例えば、ディスク故障はハードウェアの経年劣化でいずれ発生します。Ceph でディスク故障が発生した場合、他に健全なディスクが十分な数残っていれば、影響が表面化しないことが多いため、ユーザーや管理者は（積極的にシステムを監視していない場合）ディスクが故障したことに気が付かないかも知れません。さらに別のディスク故障が発生し、健全なディスクの数が足りなくなった時点でアプリケーションの I/O がハングするといった現象になります。

よって、アプリケーションへの影響を予防するためには最低限の監視が必要になります。
障害に全般に着目した場合、以下の課題があると思われます。

- Ceph に集約することによる課題
    - 障害時の影響範囲の把握
    - 障害時の対処手順の確立
- Ceph 自体の品質が安定していることによる課題
    - 何かあった場合に対するノウハウが蓄積されていない

一方、ソフトウェア障害に対しては、現象の状況やログ等の情報を参考に既知の問題かどうかをネット上で検索するところから始まり、未知の問題の場合はコアとソースコードから解析することになります。
Ceph は RADOS と呼ばれる分散オブジェクトストレージと、それを利用する各種ストレージ IF 毎のサービスという構成となっており、使用する機能範囲に応じて調査対象のコード規模は大きくなります。



テストコードや内部的に使用している Ceph 以外のコンポーネント (キーバリューストアやウエブサーバー等) を除いたコード全体の規模は Linux カーネルに含まれるすべてのファイルシステムのコード (linux-4.7/fs) の総量におよそ匹敵します。

コード規模が大きいことに加えて、本稿で紹介する検証で分かったことでもありますが、
Ceph のログはソースコード上の当該ログ出力箇所が特定しにくい状況です。
よって、Ceph のソースコードやログを日常的に見慣れていない限り、ソフトウェア障害調査は非常に困難が予想されます。

ソフトウェア障害に着目した場合、以下の課題があると思われます。

- Ceph のソースコード規模が大きいことによる課題
    - ソフトウェア障害に対応できる体制作り
        - ログの使い勝手の改善
        - ソースコード解析の参考情報の整備

2. Ceph の課題への取り組み

上述した課題のうち、本稿では「Ceph に集約することによる課題」と「Ceph のソースコード規模が大きいことによる課題」に取り組みます。
これらの課題以外の「Ceph 自体の品質が安定していることによる課題」については障害対応ノウハウの蓄積と共有が 1 つの対策と考えます。
上記課題の対策が、障害対応ノウハウとして役立つことを期待します。

2.1. Ceph に集約することによる課題への取り組み

予め、障害が発生した場合に必要な以下の情報を整理しておくことが対策となると考えます。

- 起こりうる障害
  ここでの障害はハードウェア障害を指します。
- 障害の影響
- 障害の切り分け方法
- 障害の対処方法

そのためのアプローチとしてテストを実施します。これは、与えられたシステム構成で想定される障害を「項目」として洗い出し、選択した項目について実際に障害を発生させます。
このとき、以下の情報を記録・整理します。

- 影響の観点
- 影響としての現象
- 切り分け方法
- 復旧方法
- 作業所要時間

影響の観点として、本稿では「アプリケーションの I/O への影響」と「システム管理者の操作への影響」の 2 つを設定しています。
それぞれの観点について実施したテストの具体的な内容や結果については「3. OpenStack/Ceph 異常系テスト (1)」、「5. OpenStack/Ceph 異常系テスト (2)」で述べます。

2.2. Ceph のソースコード規模が大きいことによる課題への取り組み

ソフトウェア障害発生時に障害情報として採取した Ceph のログを用いて、調査すべき Ceph ソースコード範囲を限定することを試みます。
これは以下の 3 つの作業から成ります。

- Ceph ログ行から Ceph ソースコード上のログ出力箇所 (関数) を特定する仕組みの作成
- Ceph ソースコード内の関数呼び出しの関係の洗い出しおよび処理ルートデータベース化
- 一連の Ceph ログのログ出力箇所 (関数) で処理ルートデータベースを検索し、一連のログに対応する処理ルート候補を抽出する仕組みの作成

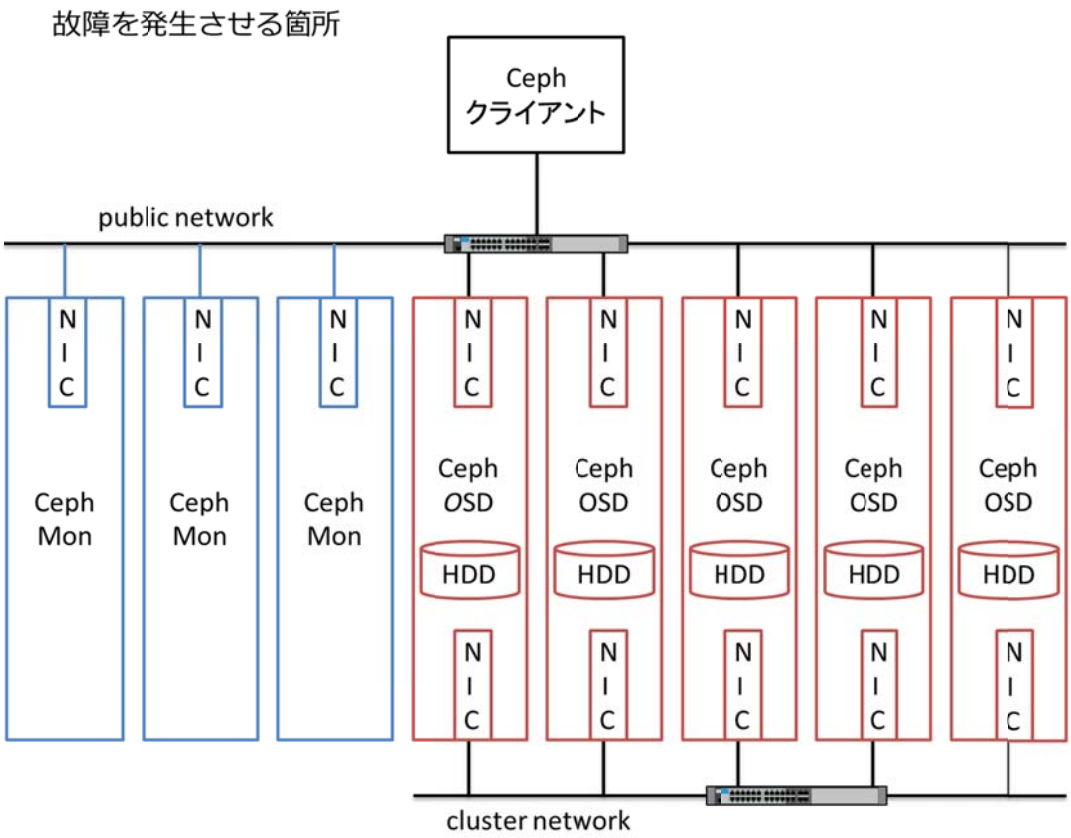
具体的な内容や結果については「4. Ceph のログ使い勝手の改善」の中で述べます。

3. OpenStack/Ceph 異常系テスト (1)

OpenStack 環境のストレージを Ceph に集約することによる課題への取り組みとして、
与えられたシステム構成で想定されるハードウェア障害が発生した際の対処に必要な情報を収集・整理します。
障害が及ぼす様々な影響のうち、ここではアプリケーションの I/O への影響に着目します。
OpenStack のインスタンス (VM) 上で OS が動作し、その上でユーザーのアプリケーションが動作します。
アプリケーションがファイル I/O を行っている状況で Ceph を構成するいずれかのハードウェアで故障を発生させます。この状況から、故障したハードウェアを交換するために必要な一連の手順を行います。故障前、故障中、復旧後を通してアプリケーションのファイル I/O の進捗を監視します。
アプリケーションのファイル I/O への影響がない (ファイル I/O がエラーにならない) ことが最も望ましい結果です。故障パターン毎にアプリケーションのファイル I/O への影響の有無や、アプリケーションのファイル I/O への影響が出る場合にはどの時間的範囲で影響が持続するのかといった情報を予め得ておくことは有用と思われます。
また、故障したハードウェアを交換するために必要な一連の手順実施に要したおよその時間も記録します。この時間は上述の影響が持続する時間的範囲に含まれる場合と含まれない場合があります。前者の場合はハードウェア故障発生時の対処の緊急性が求められる場合があります。
このテストでは特定のハードウェア障害を意図して発生させますが、実際のケースでは障害箇所の特定が必要であることは言うまでもありません。よって、故障したハードウェアを交換するために必要な一連の手順には、故障箇所の特定も含まれます。
結果的に、コマンド出力等では障害箇所の特定ができない場合がありました。また、フィールドサポートの場面としては障害箇所の特定のためのオペレーション (コマンド実行等) をタイムリーに実施することが困難な場合も想定されます。このような場合に対しては取得済みの Ceph のログからの障害箇所の特定を試みます。具体的な内容や結果については「4. Ceph のログ使い勝手の改善」の中で述べます。

3.1. システム構成

OpenStack/Ceph 異常系テスト (1) に使用したシステムの構成を説明します。
下記図に示すような構成のシステムを 7セット使用します。複数のシステムを使用する主な目的は、システム毎に異なるテスト項目を並行して実施することで、テスト全体に掛かる時間を短縮することです。



図の「Ceph クライアント」は OpenStack 環境と RBD クライアントを指します。
OpenStack 環境は、1 台のコントロールノード兼コンピュートノードと 1 台のコンピュートノードから構成されます。
OpenStack の以下のコンポーネントがストレージバックエンドに Ceph を使用します。

- OpenStack Compute (nova)
- OpenStack Block Storage Service (cinder)
- OpenStack Image Service (glance)

2つの OpenStack インスタンス (VM) を作成します。1つはイメージから起動し、1つはボリュームから起動します。各インスタンスは別々のコンピュートノードにアサインされます。各インスタンスにファイル I/O 用のボリュームを 1 つ作成し、インスタンスに接続します。
RBD クライアントは 1 台のノードです。ファイル I/O 用の RBD ボリュームを作成し、RBD クライアントにブロックデバイスとして接続します。
いずれの Ceph クライアントも Ceph のブロック I/F (RBD) を使用します。OpenStack 環境と RBD クラ

イアントではボリュームの接続方法が異なります。OpenStack 環境からは librbd ライブラリで接続し、RBD クライアントからは Linux カーネルの krbd ドライバで接続します。
本稿ではファイル I/F およびオブジェクト I/F を使用するアプリケーションへの影響の観点については扱いません。

Ceph クラスタは以下の構成です。

- Ceph MON サーバー: 3 台
- Ceph OSD サーバー: 2〜5 台、各 Ceph OSD サーバーで Ceph OSD デーモンが 1 つ動作
- ネットワーク: 2 系統

使用するソフトウェアのバージョンは以下のとおりです。

- OS: Ubuntu 14.04.2 LTS
- Ceph: hammer (v0.94.1)
- OpenStack: stable/kilo

OpenStack 環境や RBD クライアント、Ceph クラスタを構成するサーバーは全て VM です。よって OpenStack インスタンス (VM) は VM 上の VM (ネスト VM) です。

OpenStack 環境や RBD クライアント、Ceph クラスタを構成する各 VM は以下の構成です。

- CPU: 1 コア/2 ソケット
- メモリ: 8GB (OpenStack サーバー、RBD クライアント) または 1GB (Ceph MON サーバー、Ceph OSD サーバー)

上記の VM は 10 台の物理サーバーに分散して収容しています。
実際の運用で用いられるのはベアメタルサーバーで、搭載 CPU コア数や搭載メモリ量も上記 VM よりも格段に大きい場合が多いようです。

物理サーバーのネットワーク媒体は 1Gb イーサーネットを使用します。この環境では 1Gb イーサーネットを、上述の 7 セットのシステムでシェアしています。つまり、帯域性能的には 1Gb イーサーネットの 1/7 になる可能性があります。
実際の運用で用いられるのは 10Gb イーサーネットが多いようです。

物理サーバーのストレージ媒体は 3.5’ SATA HDD を使用します。各 Ceph OSD デーモンあたり 3.5’ SATA HDD 1 本の割り当てです。
実際の運用で用いられるのは SSD が多く、1 台の Ceph OSD サーバーに n 台の SSD を搭載し、これを n 個の Ceph OSD デーモンでサービスする構成のようです。

上記のように、本稿で使用するシステム構成と実際の運用で用いられるシステム構成とでは、使用するサーバースペック (CPU コア数、メモリ量) やネットワーク媒体、ストレージ媒体の構成が異なるため、本稿のテ

スト結果と実際の運用とでは性能上の傾向が大きく異なる可能性がある点に注意が必要です。
例えば Ceph OSD サーバーがダウンしたときに残りの Ceph OSD サーバーでデータ冗長性を維持するためのリバランスが動作しますが、 上記の違いにより、1 台の Ceph OSD サーバーがダウンした時に影響を受ける Ceph OSD デーモンの数が異なりますし、 リバランスに使用できるネットワーク帯域が異なるので、リバランスに掛かる時間等が大きく異なってくると思われます。

また、実際の運用では、OpenStack のコンピュートノードが Ceph OSD サーバーを兼ねる構成を採る場合があります。この場合、Ceph クラスタのリバランス処理によって発生した負荷等が OpenStack のコンピュートノード上のインスタンス (VM) に及ぼす影響や、反対に、インスタンス (VM) によって発生した負荷が Ceph クラスタの I/O 性能に及ぼす影響等も予め把握しておきたいところです。

一般に、このような観点での検証 (運用で使用するサーバーのサイジングの検証) では使用する物理サーバーのスペックも重要なパラメータとなります。本稿でのテストにはサイジングの観点は含まれていません。

3.2. テスト項目 (想定されるハードウェア障害)

上述したシステム構成において想定されるハードウェア障害のパターンをリストアップし、テスト項目とします。リストアップの方法は、はじめに故障するハードウェアのドメインやシステム構成のパラメータを「要因」としてリストアップし、次に、各ハードウェアのドメインやシステム構成のパラメータがとりうる状態や設定について「因子」としてリストアップします。最後に、各要因に設定する因子の組み合わせパターンとしてテスト項目を記述します。以下に要因とその因子を列挙します。

【故障するハードウェアのドメイン】
要因: Ceph OSD サーバーのダウン
因子: なし、1 台、2 台、3 台、4 台

要因: Ceph MON サーバーのダウン
因子: なし、1 台、2 台、3 台

要因: Ceph OSD サーバーの public NIC 故障
因子: なし、1 台、2 台、3 台、4 台

要因: Ceph MON サーバーの public NIC 故障
因子: なし、1 台、2 台、3 台

要因: Ceph OSD サーバーの cluster NIC 故障
因子: なし、1 台、2 台、3 台、4 台

要因: Ceph OSD デーモンのディスクフル
因子: なし、1 台、2 台、3 台、4 台

要因: Ceph OSD デーモンのディスク故障
因子: なし、1 台、2 台、3 台、4 台

要因: public network
因子: 正常、スイッチ故障

要因: cluster network
因子: 正常、スイッチ故障

【システム構成のパラメータ】
要因: Ceph pool size (=データ冗長度)
因子: 2, 3, 4, 5

要因: Ceph pool min_size (=I/O 継続に必要なデータ冗長度)
因子: 2

要因: スペア Ceph OSD サーバー (Ceph OSD サーバーダウン時等)
因子: あり、なし

要因: Ceph OSD サーバーの交換 (Ceph OSD サーバーの NIC 故障時等)
因子: 交換しない、交換する

要因: Ceph MON サーバーの交換 (Ceph MON サーバーの NIC 故障時等)
因子: 交換しない、交換する

要因: Ceph OSD デーモンあたりの PG 数
因子: mon_pg_warn_min_per_osd (30) 以上、mon_pg_warn_max_per_osd (300) 以下
mon_pg_warn_min_per_osd (30) より小さい
mon_pg_warn_max_per_osd (300) より大きい

具体的なテスト項目については「3.4 テスト結果」に記述します。

3.3. テスト方法

以下の方法で、OpenStack から Ceph へのワークロードが継続的にある状態をつくります。
各システムに 2 つのインスタンス (VM) が置かれています。1 つはイメージから起動、1 つはボリュームから起動しています。各インスタンスにはファイル I/O 用のボリュームが 1 つ追加されています。ファイル I/O 用のボリュームにファイルシステムを構築し、インスタンスからマウントします。各インスタンス上に I/O プロセスを起動します。I/O プロセスは、1 秒間のスリープと上記ファイルシステム上に 1 個の新規ファイルを作成する処理を繰り返します。

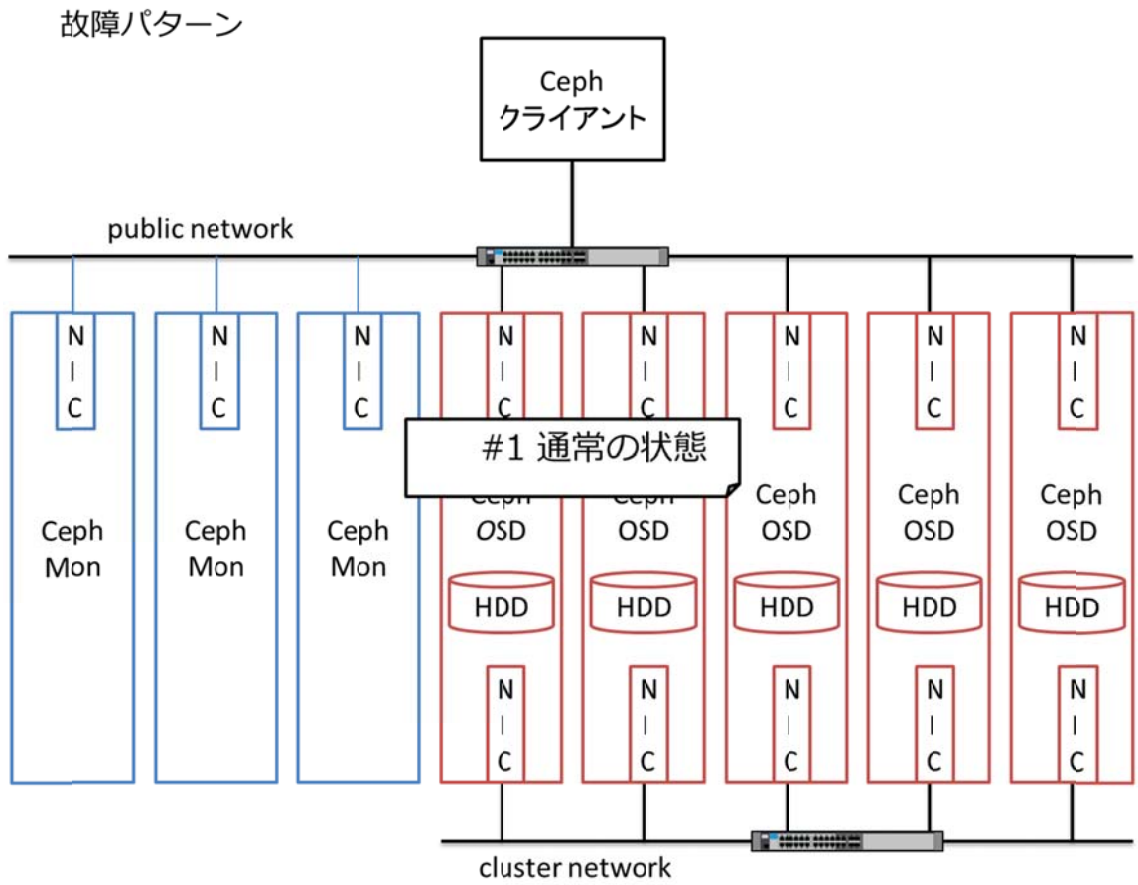
OpenStack から Ceph へのワークロードの他に、RBD クライアントから Ceph へのワークロードも発生させます。上述したように、OpenStack 環境と RBD クライアントではボリュームの接続方法が異なります。OpenStack 環境からは librbd ライブラリで接続し、RBD クライアントからは Linux カーネルの krbd ドライバで接続します。また、両者でファイル I/O の仕方を変えます。後述するように、RBD クライアントからは DirectIO を使用します。

各システムに 1 つの RBD クライアントが置かれています。RBD クライアントにはファイル I/O 用の RBD ボリュームが 1 つ接続されています。ファイル I/O 用の RBD ボリュームにファイルシステムを構築し、RBD クライアントからマウント (sync オプション指定) します。RBD クライアント上に I/O プロセスを起動します。I/O プロセスは、1 秒間のスリープと上記ボリューム上に 1 個の新規ファイルを作成して 4KB のデータを write (direct I/O) する処理を繰り返します。

次に、この状態でテスト項目に記述したハードウェア障害を発生させます。ハードウェア障害の発生方法については説明を省略します。
何らかの理由で新規ファイルが作成できない場合、I/O プロセスは終了します。各テスト項目実施を通して当該プロセスが終了しないことをもって運用が継続できている状態とみなします。Ceph へのワークロードが (I/O がブロックされる等で) 停止と再開を繰り返している状態も運用が継続できている状態の一部とみなします。

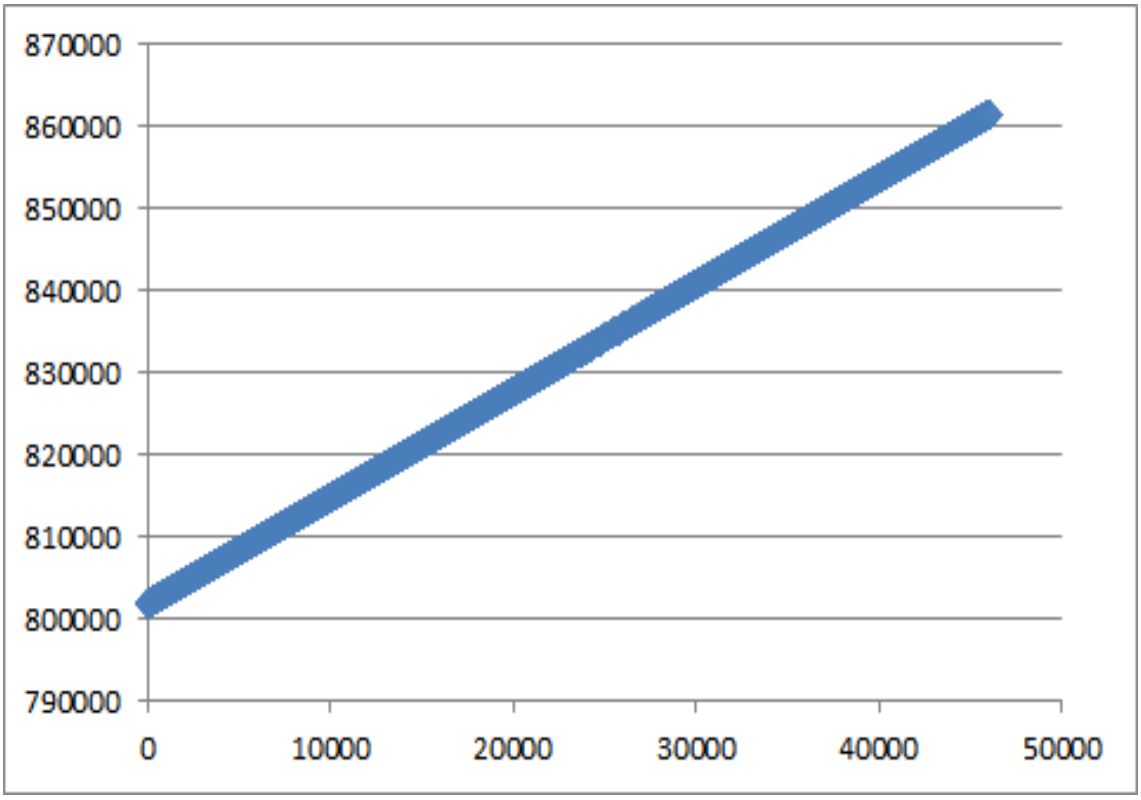
3.4. テスト結果

具体的なテスト項目とその結果について説明します。
始めに、各テスト項目を表現した図の見方を説明します。テスト項目は、故障するハードウェアのドメインの状態とシステム構成のパラメータの値のパターンです。



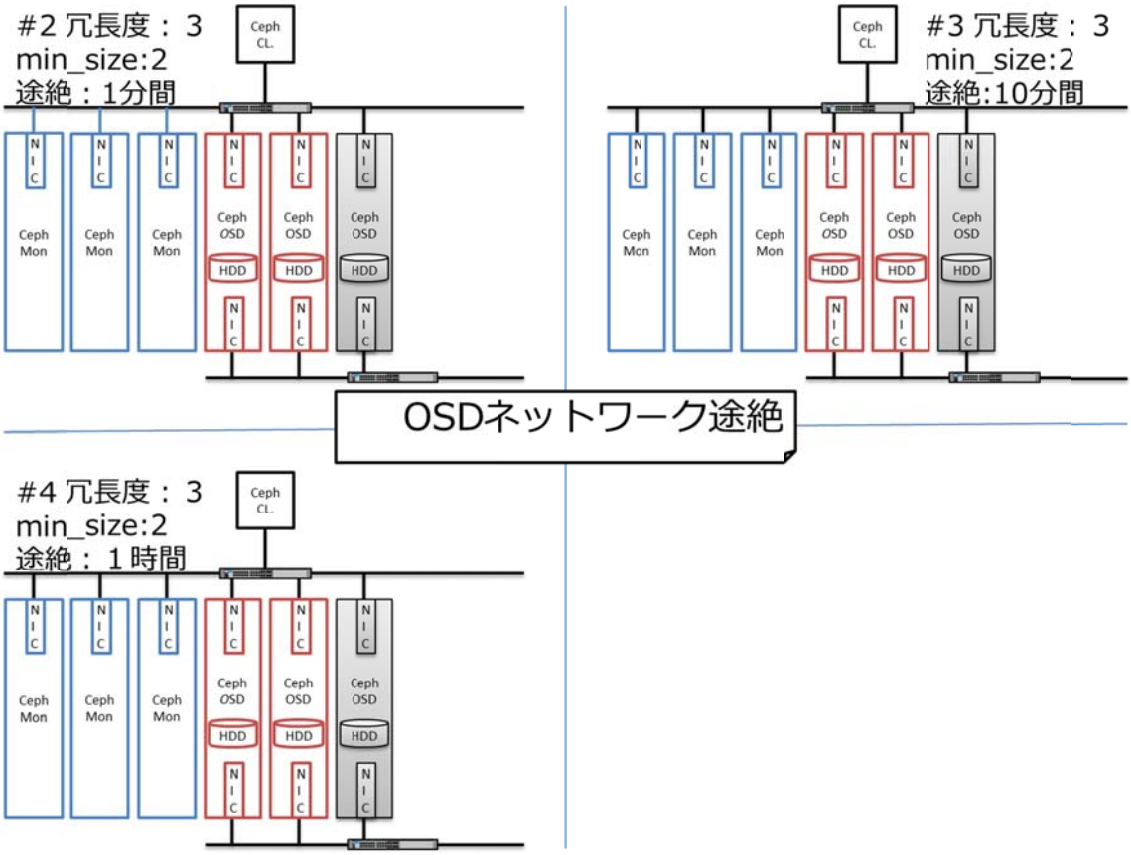
この図はテスト項目 #1 を表しており、全てのテスト項目の基準となります。
この図では 3 台の Ceph MON サーバーと 5 台の Ceph OSD サーバーがあり、全てのサーバーが public network に接続されており、全ての Ceph OSD サーバーが cluster network に接続されている状態を表しています。
故障するハードウェアのドメインは灰色で示されます。この図では灰色のドメインは存在せず、すなわち、通常の状態を表します。
システム構成のパラメータについては、注目すべきものについて余白に記載します。この図では何も記載されていません。

つぎに下記の図の見方を説明します。



OpenStack/Ceph 異常系テスト (1) では、障害が及ぼす様々な影響のうち、アプリケーションの I/O への影響に着目します。
上記の図は、障害 (＝テスト項目) 毎に示されるアプリケーション I/O の進捗を表します。縦軸は単位時間を表し、横軸はファイル I/O 用のボリュームに作成されたファイルの数を表します。データは、RBD クライアント上で DirectIO を使用してワークロードを生成している I/O プロセスでの結果を使用しています。
テスト項目 #1 は通常の状態での結果です。時間の経過におよそ比例して作成されるファイル数が増加しています。このグラフは、ファイルを 1 つ作成する毎に点をプロットしています。 (複数の点を結んだ直線でないことに注意してください。)

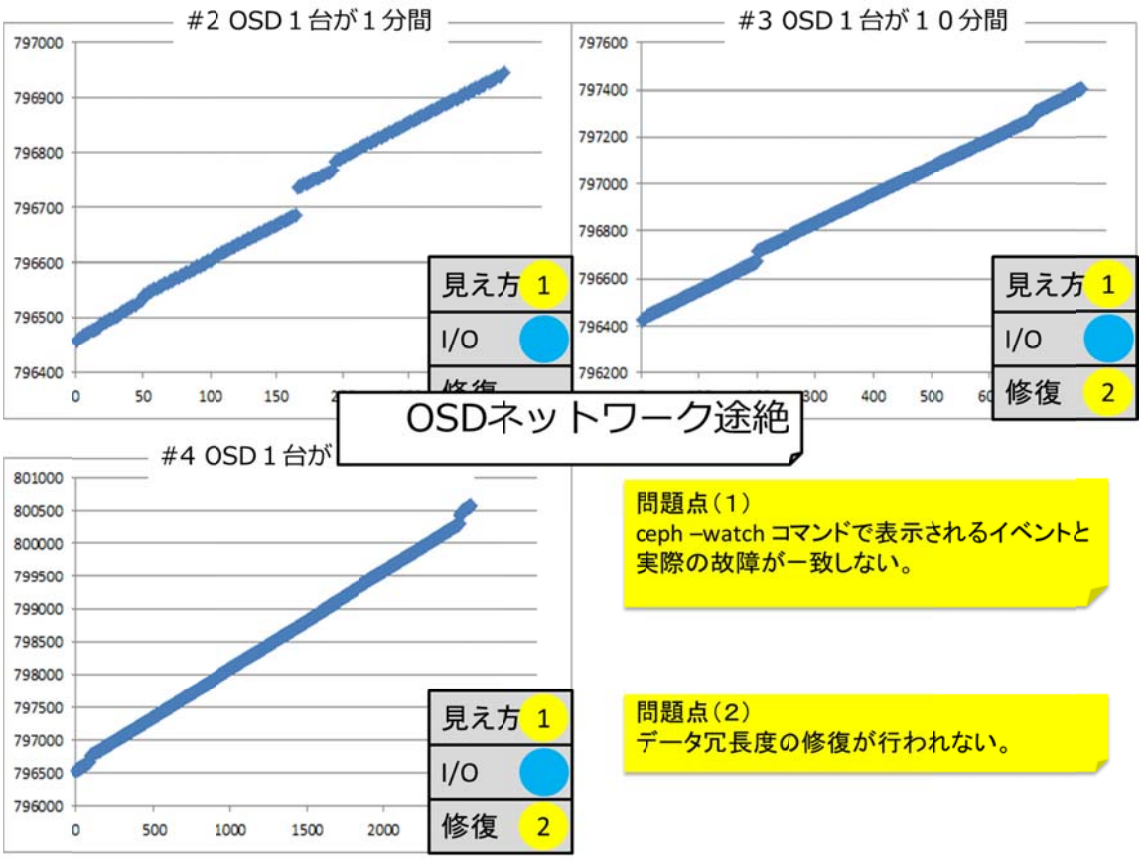
もし、ファイル I/O でエラーが発生すると I/O プロセスは終了するのでその時点以降、ファイル数はプロットされません。もし、ファイル I/O でハングが発生すると、I/O プロセスもハングするので、ハングしている間はファイル数がプロットされません。ハングが解けた時点で再びファイル数がプロットされますので、点の集まりはハングが持続した長さだけ上下に分断されるはずです。また、ファイル I/O の進捗が滞ると点の集まりの傾きが急になり、逆に進捗が捗ると傾きが緩やかになるはずです。



テスト項目 #2 から #4 です。

ここでは Ceph OSD サーバーの 1 台でネットワーク途絶が発生しています。各サーバーは 2 つの NIC が両方とも灰色になっていますので、public network と cluster network の両方のネットワークから途絶しています。

システム構成のパラメータについては、注目すべきものについて余白に記載します。ネットワーク途絶の状態が持続した時間が余白に記載されています。また、冗長度と min_size の値がそれぞれ記載されています。冗長度は pool size の値で、プールのデータ冗長度を指します。Ceph OSD デーモンは 3 台で、pool size が 3 なので、全てのプールはこれら 3 台の Ceph OSD でデータ冗長化されているはずです。よって、1 台がネットワーク途絶の間は全てのプールでデータ冗長度に満たない状態です。min_size は pool min_size の値で、I/O を継続するデータ冗長度の設定です。健全な Ceph OSD デーモンが 2 台残っている状況で min_size が 2 なので全てのプールで I/O は継続されるはずです。



テスト項目 #2 から #4 です。

いずれも点の集まりが不連続な箇所があるので、一時的な I/O ハングが発生していたと思われます。

各図の右下に示されている表示の見方について説明します。

「見え方」はコマンドの表示に発生したハードウェア障害を特定するような情報が含まれていたかどうかを指します。青色のマークは、コマンドがハードウェア障害を特定するような情報を表示したことを表します。黄色のマークは、コマンドが何等かの表示を行ったが、ハードウェア障害を直接示すものではなかったことを表します。赤色のマークは、コマンドの表示がなかったことを表します。

「I/O」は障害が及ぼす様々な影響のうち、アプリケーションの I/O への影響が許容できるものかどうかを示します。青色のマークは、影響なしまたは影響が軽微であったことを表します。黄色のマークは、影響があったものの I/O は 継続されたこと表します。赤色のマークは 、I/O が継続されなかったことを表します。

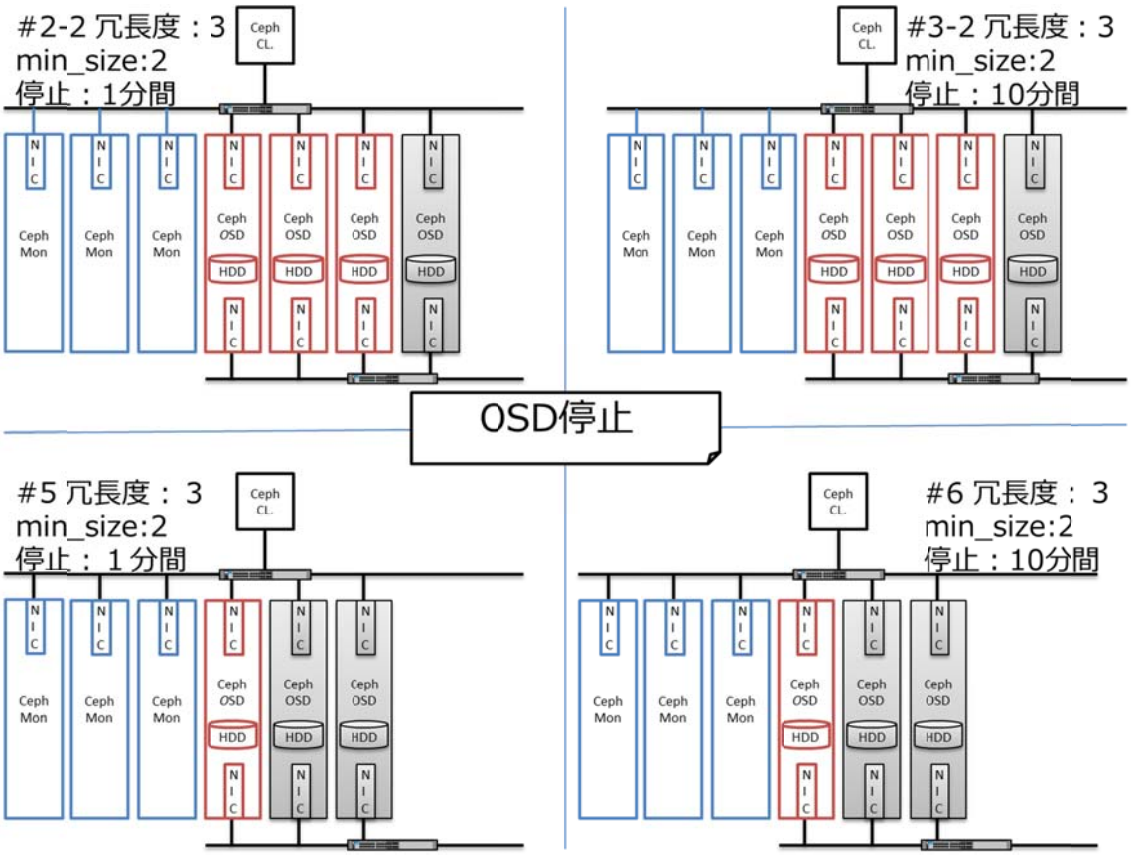
「修復」はデータ冗長度が維持されたかどうかを示します。マークなしは、障害の持続時間が短く、データ冗長度の修復が開始よりも前に障害が復旧したことを表します。青色のマークは、データ冗長度が維持されたことを表します。黄色のマークは、データ冗長度が下がったものの、データが失われることはなく、ハードウェア障害を取り除くことでデータ冗長度が維持されることを表します。赤色のマークは、データが失われたことを表します。

青色、黄色、赤色のマークに数値が記載されている場合は具体的な記述へのポインタです。上記の図の場合、問題点 (1) 、問題点 (2) を指しています。

問題点 (1) に関してより具体的には Ceph OSD サーバーのネットワーク途絶ではなく Ceph OSD サーバーのダウンとして表示されたため、コマンドの表示と実際の障害が一致していなかったことを示しています。

問題点 (2) に関しては、スペアの Ceph OSD デーモンがないためデータ冗長度の修復が開始されなかっ

たが、ハードウェア障害を取り除くことでデータ冗長度が維持されたことを示しています。スペアの Ceph OSD を用意しておくことで回避可能な問題です。



テスト項目 #2-2 から #6 です。

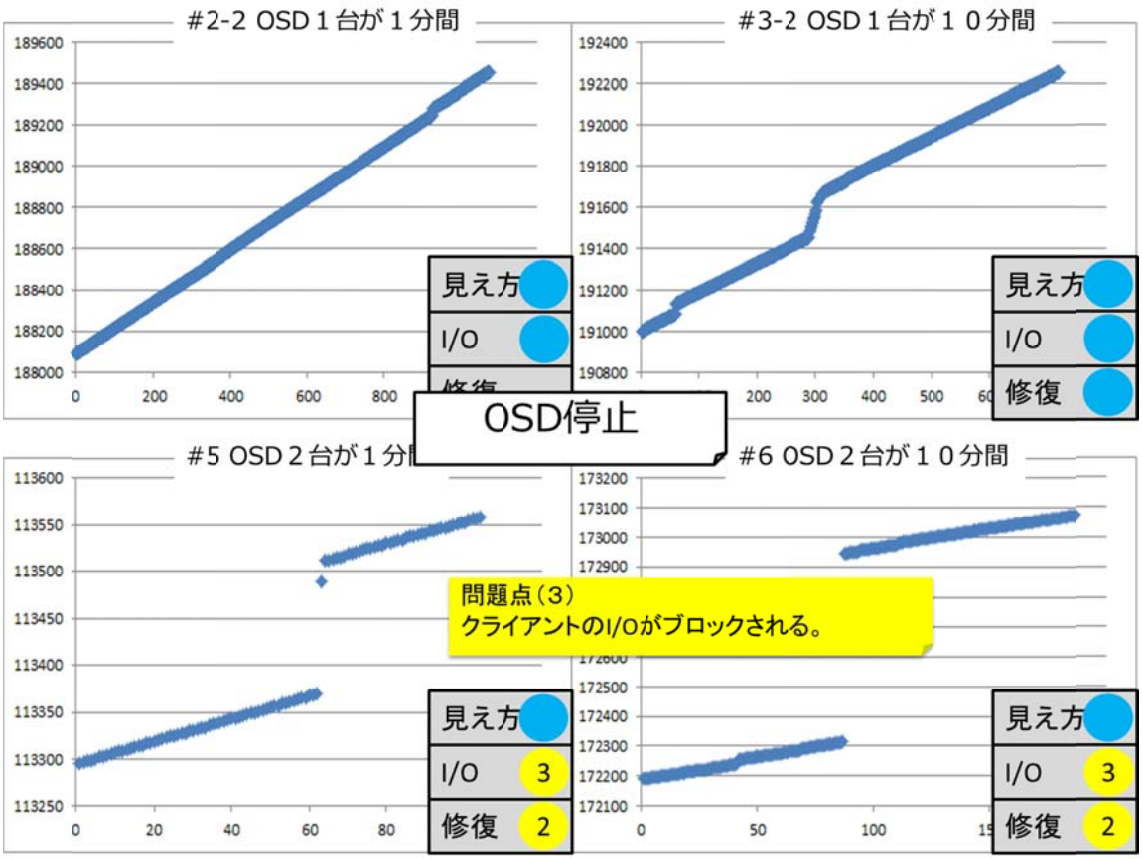
ここでは Ceph OSD サーバーの 1 台または 2 台でサーバーダウンが発生しています。

余白に記載されている「停止： 1 分間」、「停止： 10 分間」は、サーバーダウンの状態がそれぞれ 1 分間以上、10 分間以上持続したことを表します。

「冗長度」はいずれのテスト項目も 3 なので、プールはいずれか 3 台の Ceph OSD デーモンで構成されています。

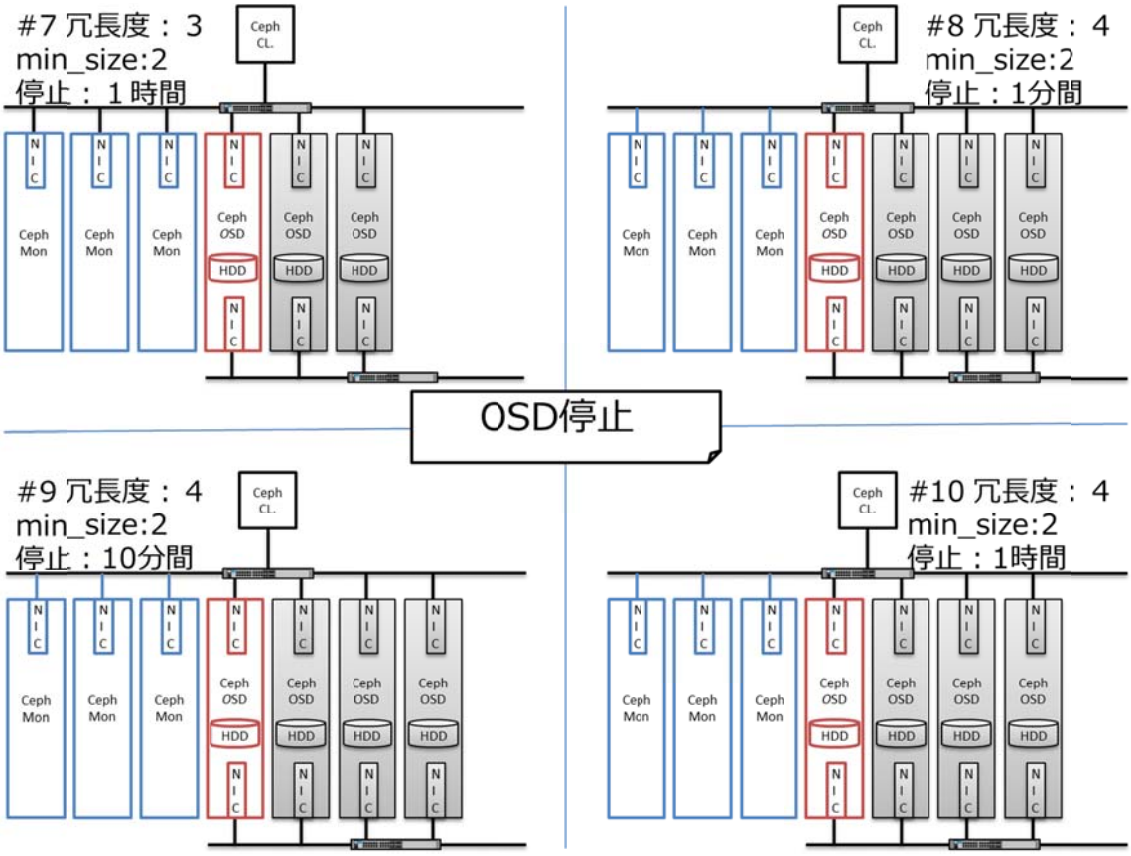
「min_size」はいずれのテスト項目も 2 なので、プールを構成する 3 台の Ceph OSD デーモンのうち 1 台がダウンしても I/O を継続します。

よって、サーバーダウン中、テスト項目 #2-2 と #3-2 では I/O が継続され、#5 と #6 では I/O がブロックされるはずです。



グラフを確認すると、サーバーダウン中、テスト項目 #2-2 と #3-2 では I/O が継続されていたこと、テスト項目 #5 と #6 では I/O がブロックされていたことが分かります。テスト項目 #5 と #6 の I/O プロセスはいずれも終了していないので、ダウンしたサーバーが復旧した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。サーバーダウン中、テスト項目 #3-2 では I/O の進捗が遅くなっていたようです。これは、サーバーダウン中にデータ冗長度の修復処理が動作した影響の I/O 性能劣化と思われます。テスト項目 #2-2 で I/O 性能劣化が見られないのは、サーバーダウンの状態が持続する時間 (1 分間以上) が短く、データ冗長度の修復処理が開始される前にダウンしたサーバーが復旧したためと思われます。

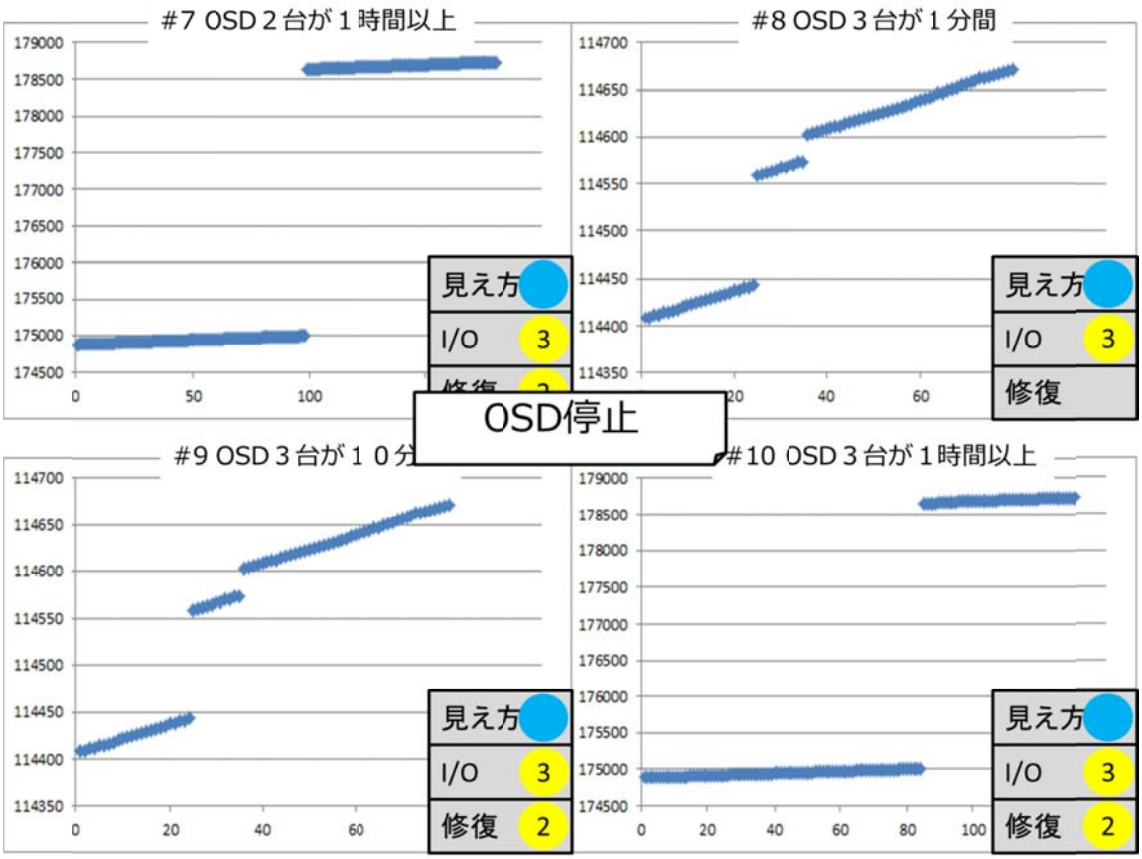
各図の右下に示されている表示については最後にまとめることにして、テスト項目毎の紹介は省略します。



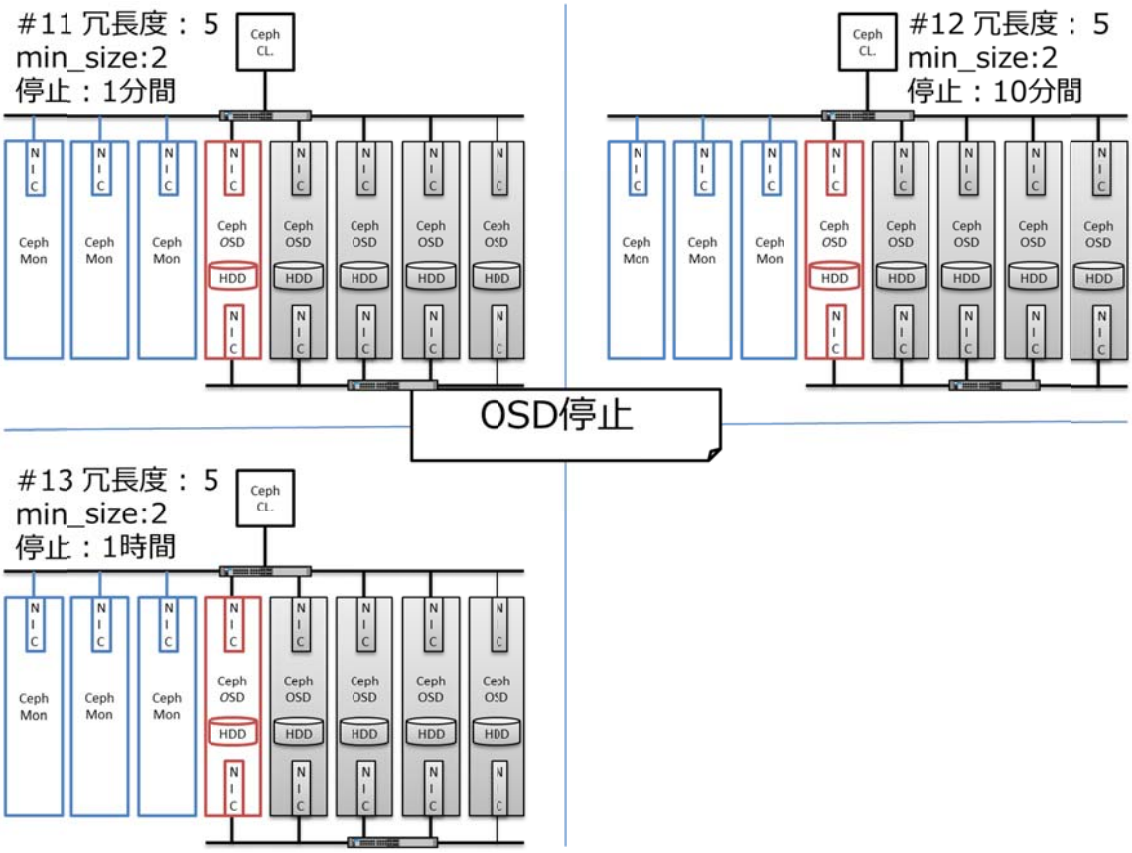
Ceph OSD サーバーダウンの項目が続きます。テスト項目 #7 から #10 です。

ここでは Ceph OSD サーバーの 2 台または 3 台でサーバーダウンが発生しています。先程のテスト項目に比べて、データ冗長度とダウンする Ceph OSD サーバーの数が増えています。

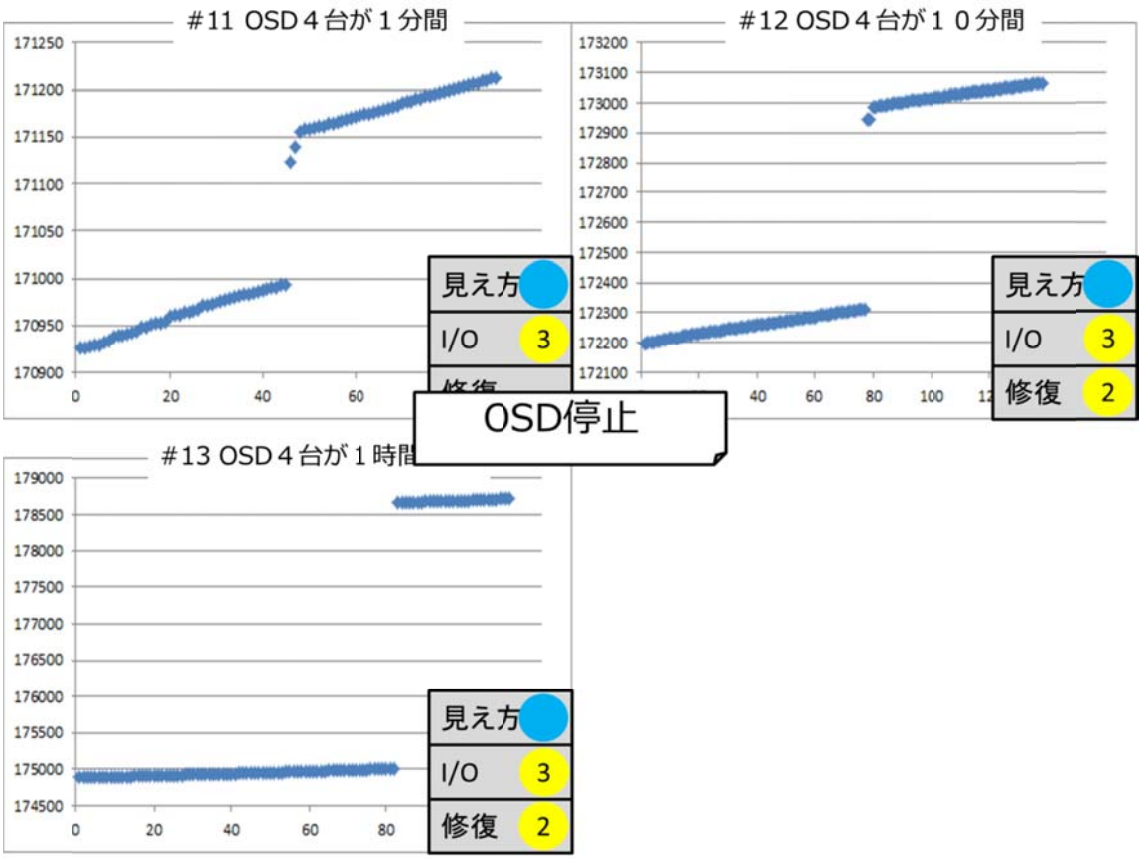
「min_size」はいずれのテスト項目も 2 に対し、サーバーダウン中の健全な Ceph OSD デーモンが 1 台しか残らない状況です。よって、サーバーダウン中、いずれの項目も I/O がブロックされるはずです。



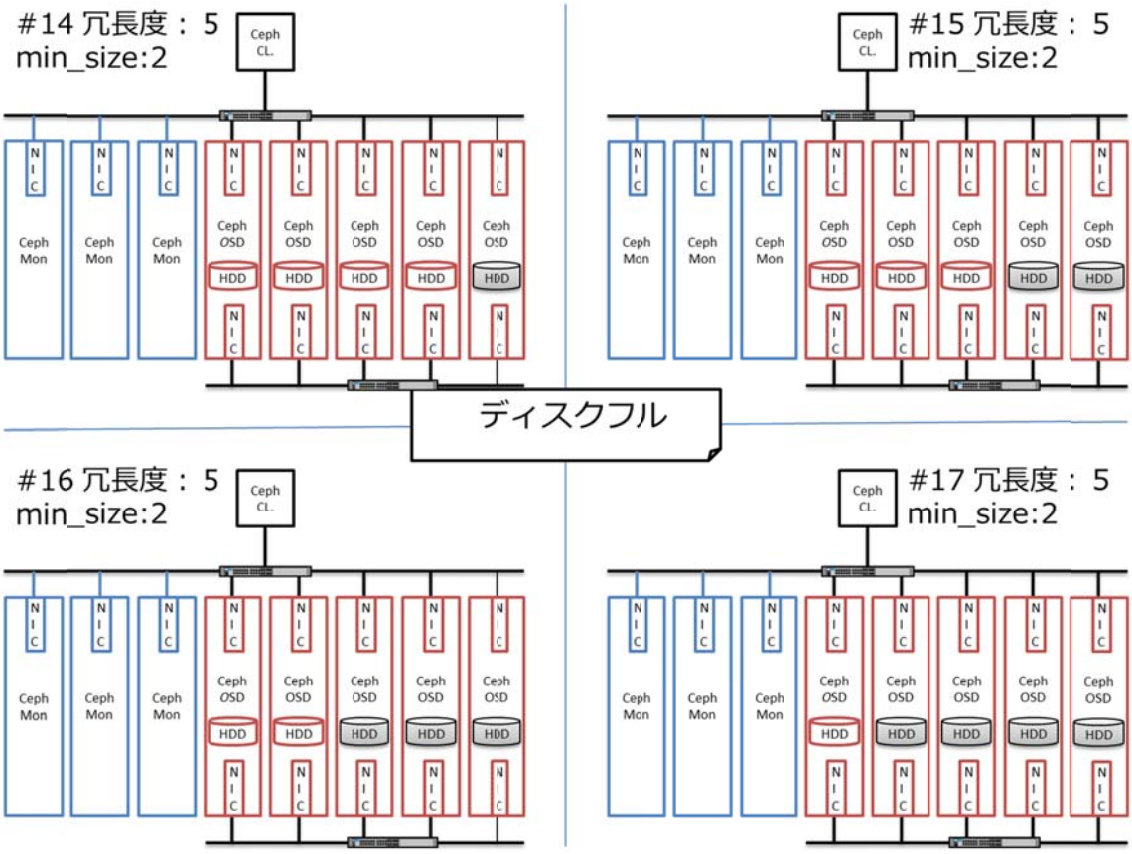
テスト項目 #7 から #10 ではサーバーダウン中 I/O がブロックされていたことが分かります。
I/O プロセスはいずれも終了していないので、ダウンしたサーバーが復旧した時点で I/O のブロックが解け、
I/O エラーは発生していないことが分かります。



Ceph OSD サーバーダウンの項目が続きます。テスト項目#11 から#13 です。
ここでは Ceph OSD サーバーの 4 台でサーバーダウンが発生しています。先程のテスト項目に比べて、さらにデータ冗長度とダウンする Ceph OSD サ ーバーの数が増えています。
「min_size」はいずれのテスト項目も 2 に対し、サーバーダウン中の健全な Ceph OSD デーモンが 1 台しか残らない点は同じです。

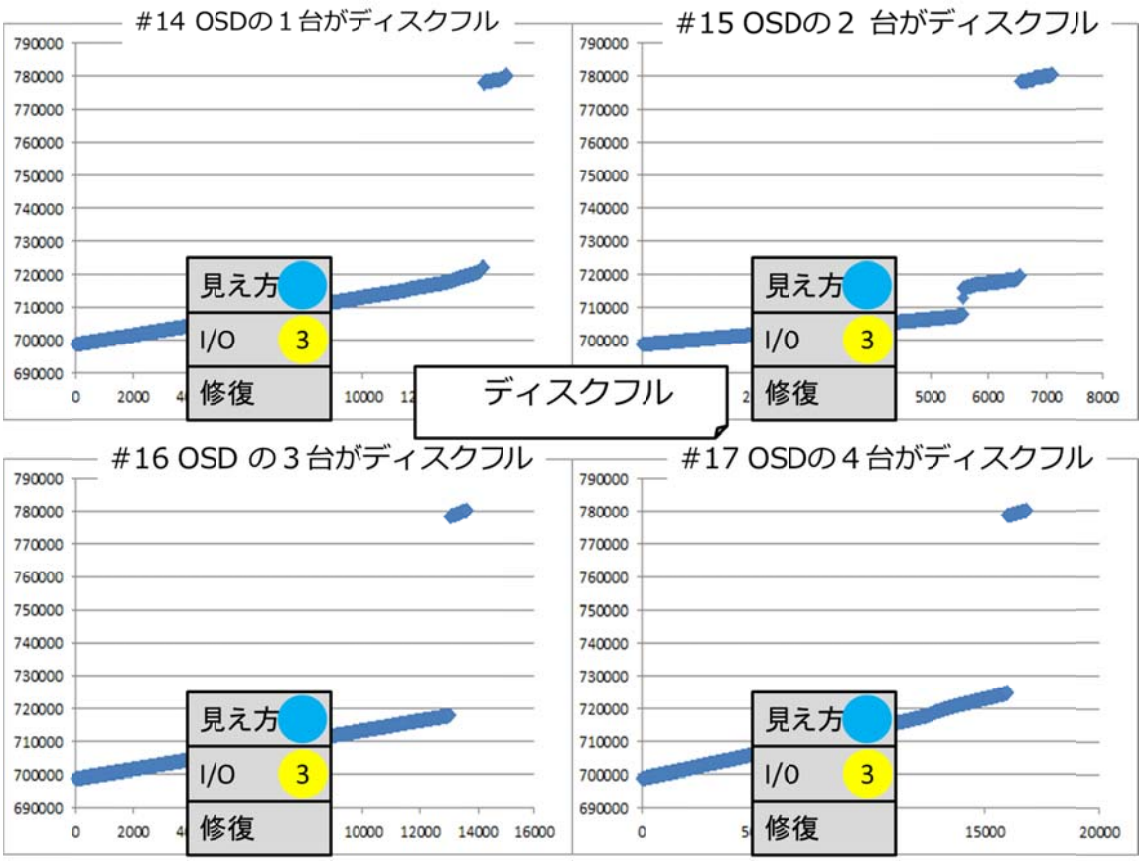


テスト項目 #5 から #10 と同様に、サーバーダウン中 I/O がブロックされていたことが分かります。ここまでの項目が Ceph OSD サーバーダウンに関連するものです。

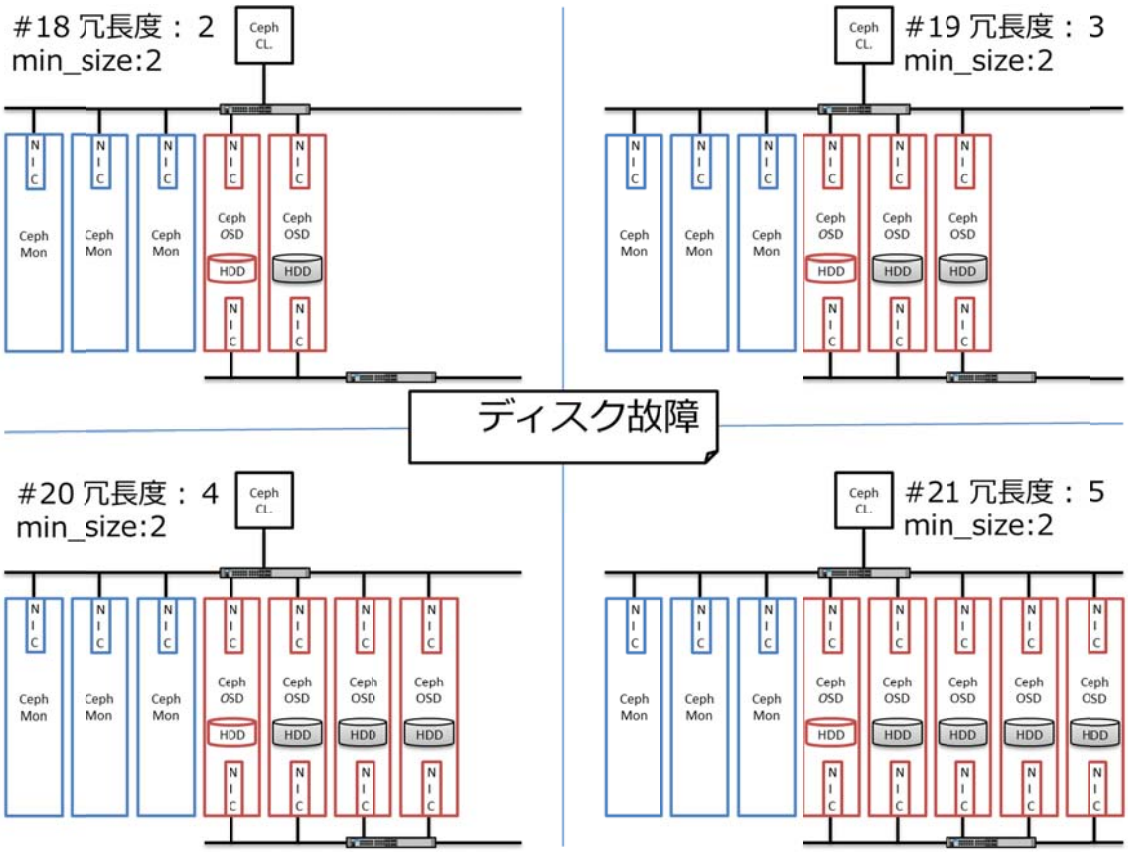


ここからの項目は Ceph OSD のディスクフルに関するものです。テスト項目 #14 から #17 です。
ここでは、データ冗長度の設定が 5 の環境で、1 台から 4 台の Ceph OSD デーモンでディスクフルが発生しています。ディスクフルの状態は、Ceph OSD デーモン毎のディスク使用率の値が full ratio パラメータの値に達した状態を指します。full ratio パラメータの値は設定で変更可能です。full ratio パラメータの他に near full ratio パラメータがあり、full ratio よりも低い値を設定します。
ある Ceph OSD デーモンのディスク使用率が near full ratio に達した時点でワーニングメッセージが出力されるようになり、full ratio に達した時点で I/O がブロックされます。

なお、1 台の Ceph OSD がディスクフルの状態となった時点で既存オブジェクトの伸長や新規オブジェクトの作成ができなくなるため、2 台以上の Ceph OSD がディスクフルの状態は full ratio の設定を変更することで作成します。すなわち、まずデータ冗長度の設定が 1 のダミープールを複数作成します。データ冗長度の設定が 1 の場合、プールを構成する PG は 1 台の Ceph OSD で構成されます。次に、Ceph OSD を選択し、当該 Ceph OSD で構成された PG で構成されたダミープールを選択し、当該ダミープールにダミーオブジェクトを配置することで各 Ceph OSD のディスク使用率が段階的に異なる状態を作成します。
次に full ratio の値の設定を変更することで意図した台数の Ceph OSD のみがディスクフルの状態を作成します。



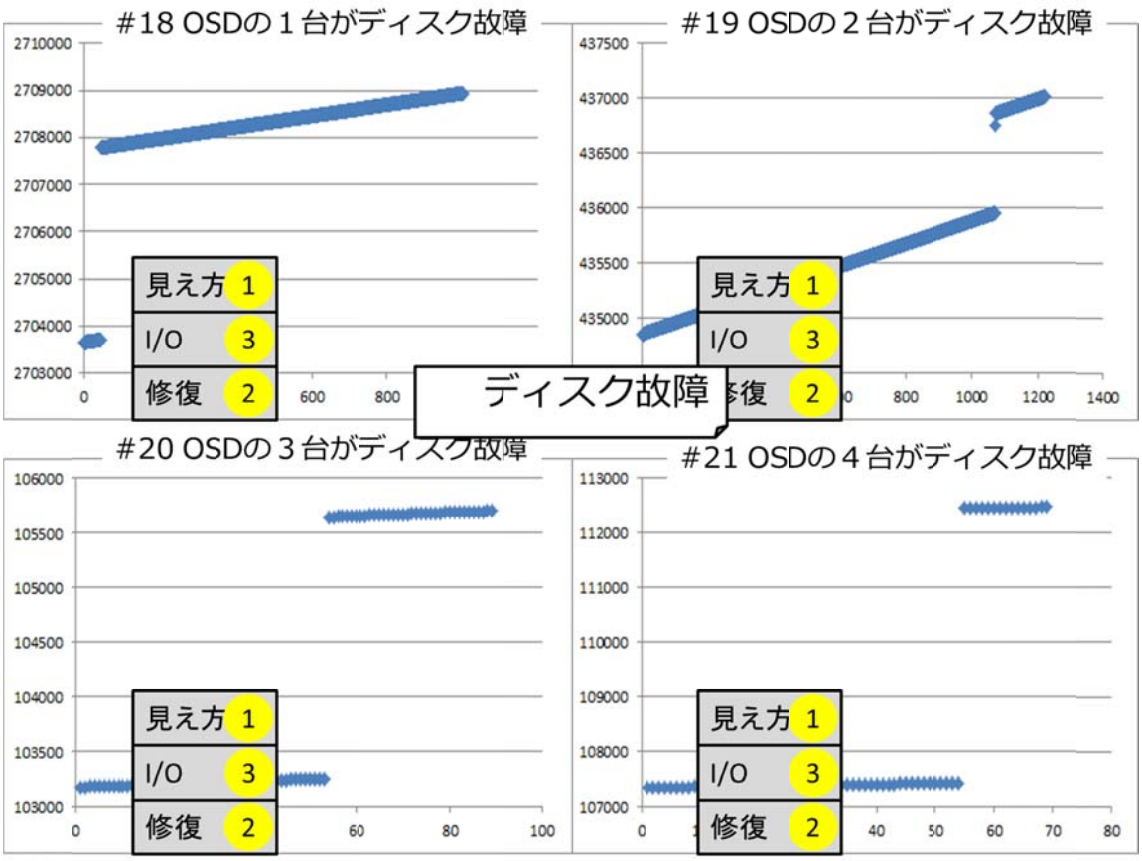
グラフを確認すると、テスト項目 #14 から #17 は I/O がブロックされていたことが分かります。I/O プロセスはいずれも終了していないので、ディスクフルが解消した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。



(図)

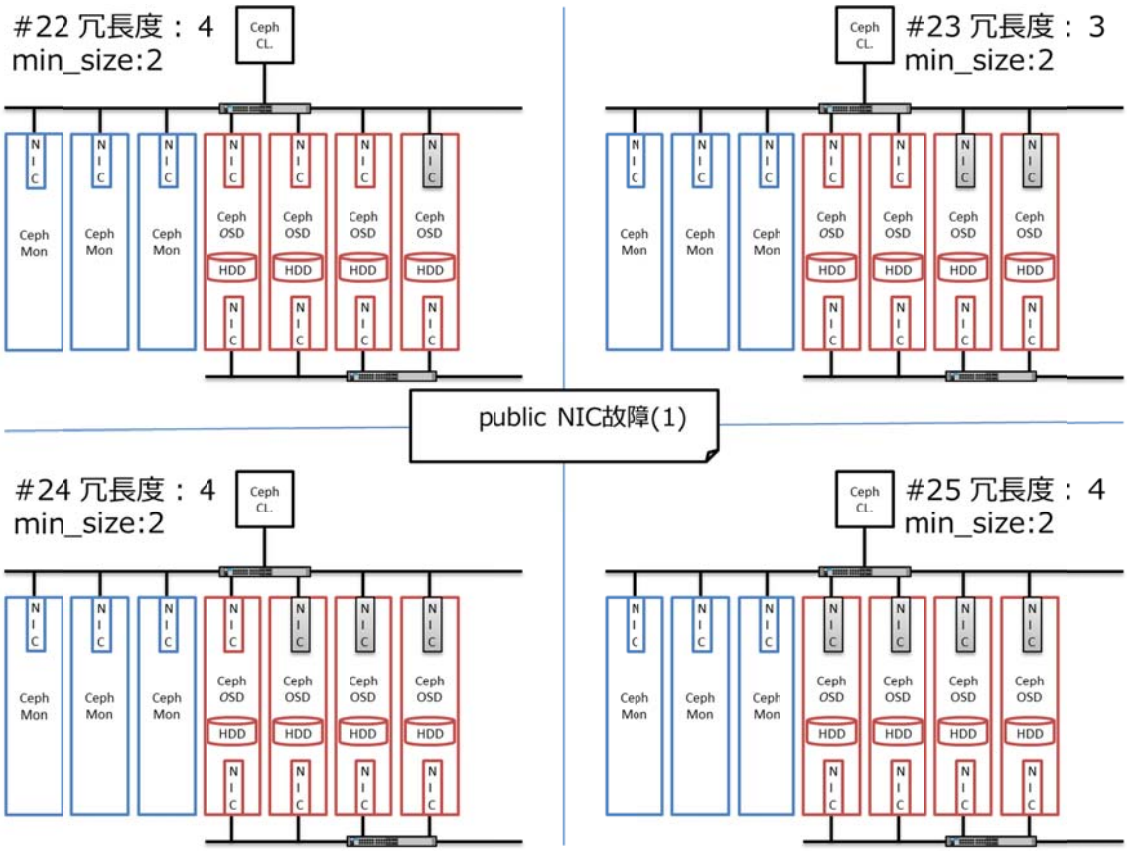
ここからの項目は Ceph OSD のディスク故障に関するものです。テスト項目#18 から#21 です。

ここでは、データ冗長度の設定が 2 から 5 の環境で、1 台から 4 台の Ceph OSD デーモンでディスク故障が発生しています。「min_size」はいずれのテスト項目も 2 に対し、ディスク故障中の健全な Ceph OSD デーモンが 1 台しか残らない状況です。よって、ディスク故障中、いずれの項目も I/O がブロックされるはずです。



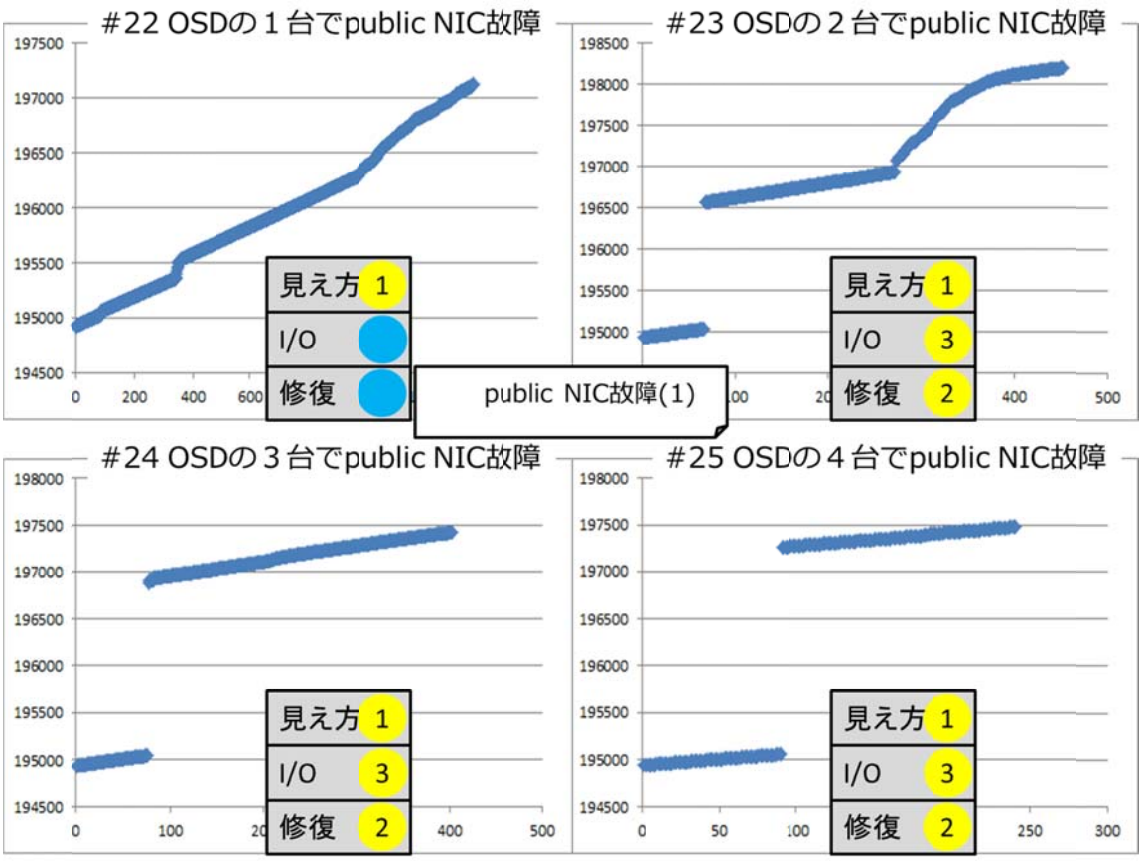
テスト項目 #18 から #21 ではディスク故障中 I/O がブロックされていたことが分かります。I/O プロセスはいずれも終了していないので、ディスクが故障したサーバーが復旧した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。

ディスクが故障したサーバーの復旧は、当該 Ceph OSD を一旦削除した後、故障したディスクを交換し、新規 Ceph OSD として再度追加することで行っています。



ここからの項目は Ceph OSD の public network 側の NIC 故障に関するものです。テスト項目 #22 から #25 です。

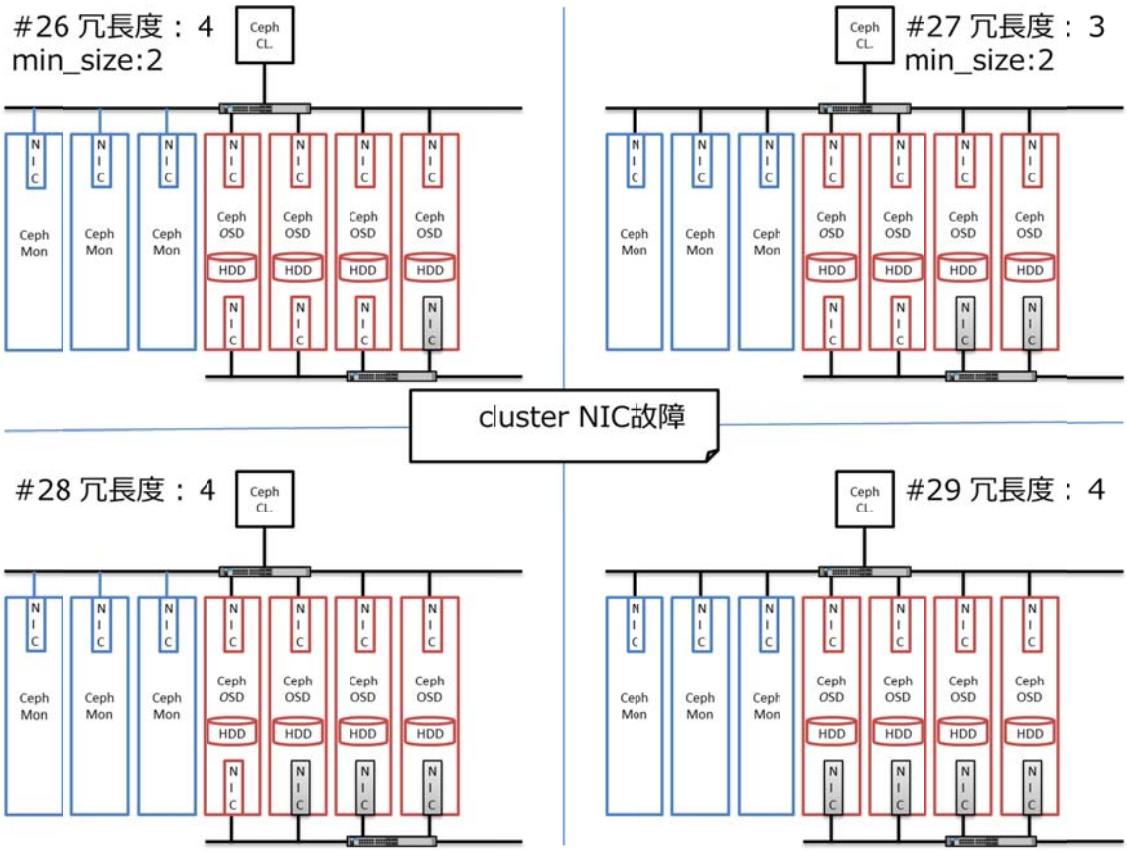
ここでは、データ冗長度の設定が 3 または 4 の環境で、1 台から 4 台の Ceph OSD デーモンで NIC 故障が発生しています。「min_size」はいずれのテスト項目も 2 に対し、テスト項目 #23 から #25 では健全な Ceph OSD が 2 台を下回る状況です。よって、NIC 故障中、いずれの項目も I/O がブロックされるはずです。



テスト項目 #23 から #25 では NIC 故障中 I/O がブロックされていたことが分かります。I/O プロセスはいずれも終了していないので、NIC が故障したサーバーが復旧した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。

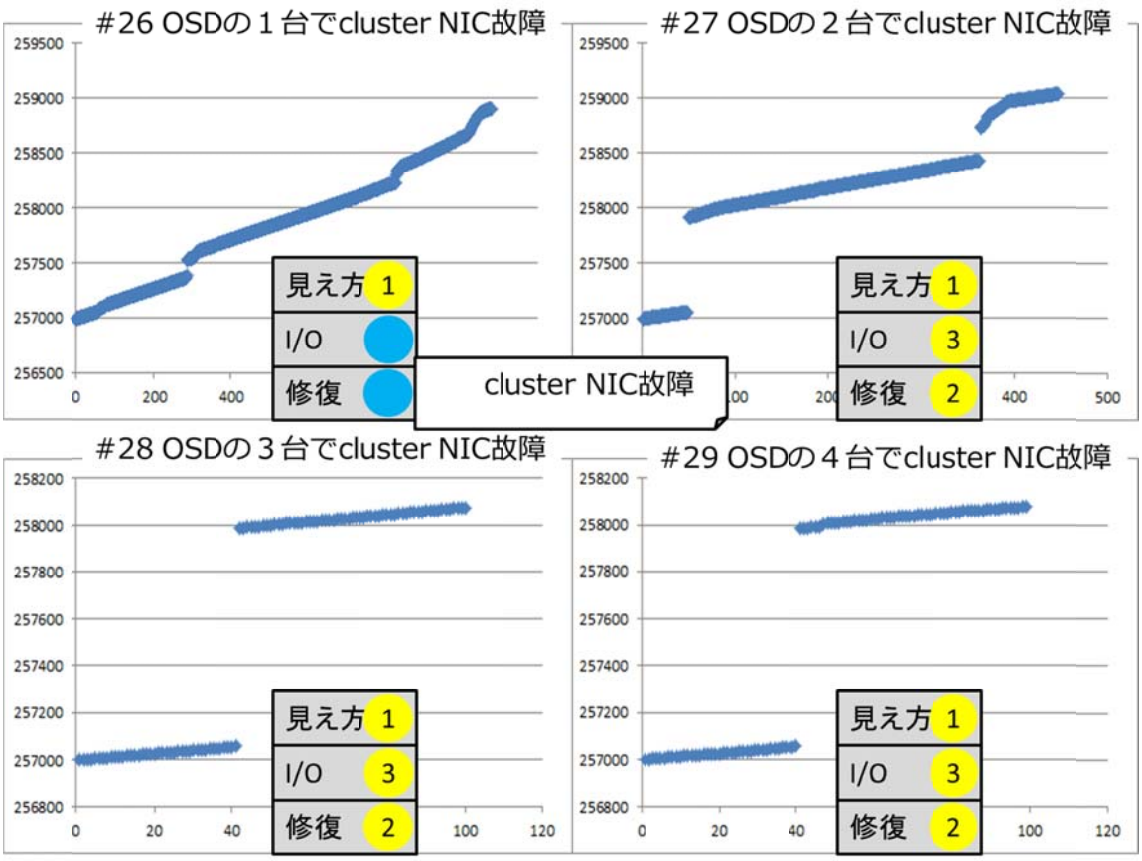
NIC が故障したサーバーの復旧は、テスト項目 #22 と #23 は当該 Ceph OSD を一旦削除した後、故障した NIC を交換し、新規 Ceph OSD として再度追加することで行っています。

テスト項目 #22 と #23 については当該 Ceph OSD をシャットダウンした後、故障した NIC を交換し、再起動することでも復旧可能です。テスト項目#24 と#25 は当該 Ceph OSD をシャットダウンした後、故障した NIC を交換し、再起動することで行っています。テスト項目 #24 については当該 Ceph OSD を一旦削除した後、故障した NIC を交換し、新規 Ceph OSD として再度追加することでも復旧可能です。



ここからの項目は Ceph OSD の cluster network 側の NIC 故障に関するものです。テスト項目 #26 から #29 です。

ここでは、データ冗長度の設定が 3 または 4 の環境で、1 台から 4 台の Ceph OSD デーモンで NIC 故障が発生しています。「min_size」はいずれのテスト項目も 2 に対し、テスト項目 #27 から #29 では健全な Ceph OSD が 2 台を下回る状況です。よって、NIC 故障中、いずれの項目も I/O がブロックされるはずです。

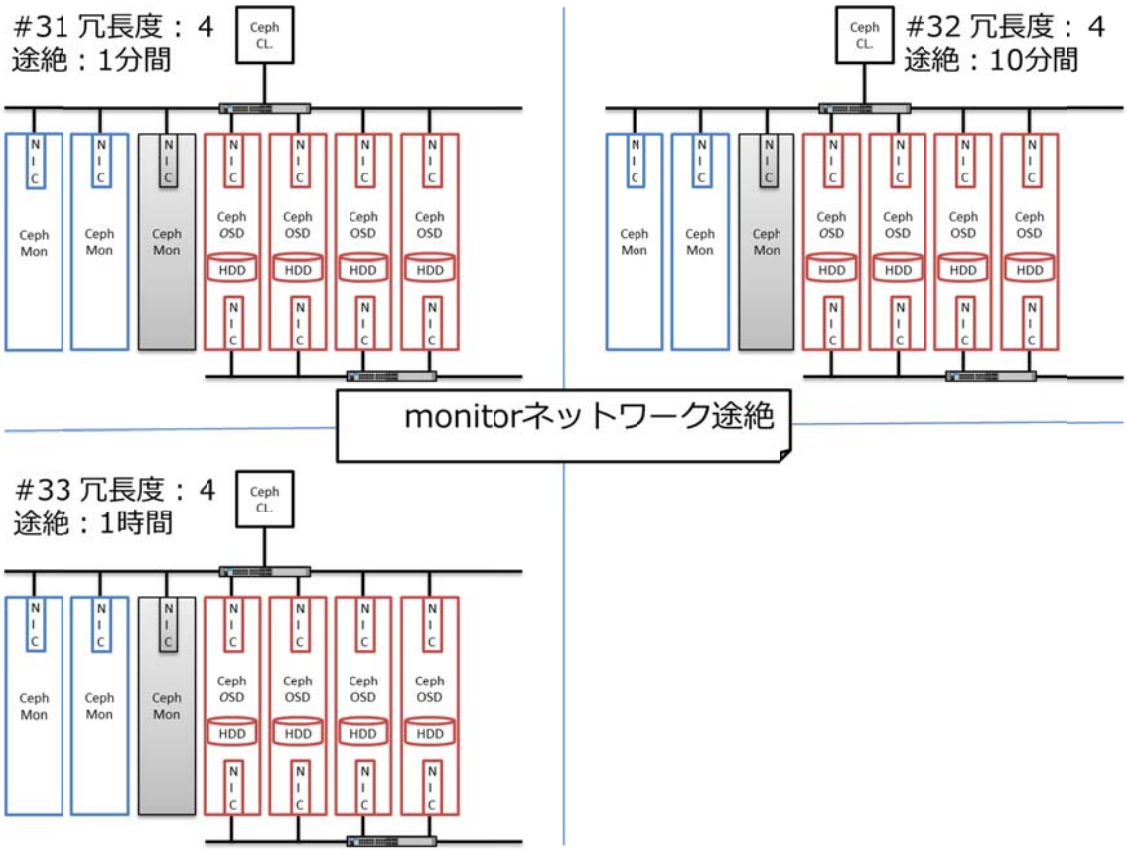


テスト項目 #27 から #29 では NIC 故障中 I/O がブロックされていたことが分かります。I/O プロセスはいずれも終了していないので、NIC が故障したサーバーが復旧した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。

NIC が故障したサーバーの復旧は、テスト項目 #26 と# 27 は当該 Ceph OSD を一旦削除した後、故障した NIC を交換し、新規 Ceph OSD として再度追加することで行っています。

テスト項目 #26 と #27 はついては当該 Ceph OSD をシャットダウンした後、故障した NIC を交換し、再起動することでも復旧可能です。テスト項目 #28 と #29 については当該 Ceph OSD をシャットダウンした後、故障した NIC を交換し、再起動することで行っています。

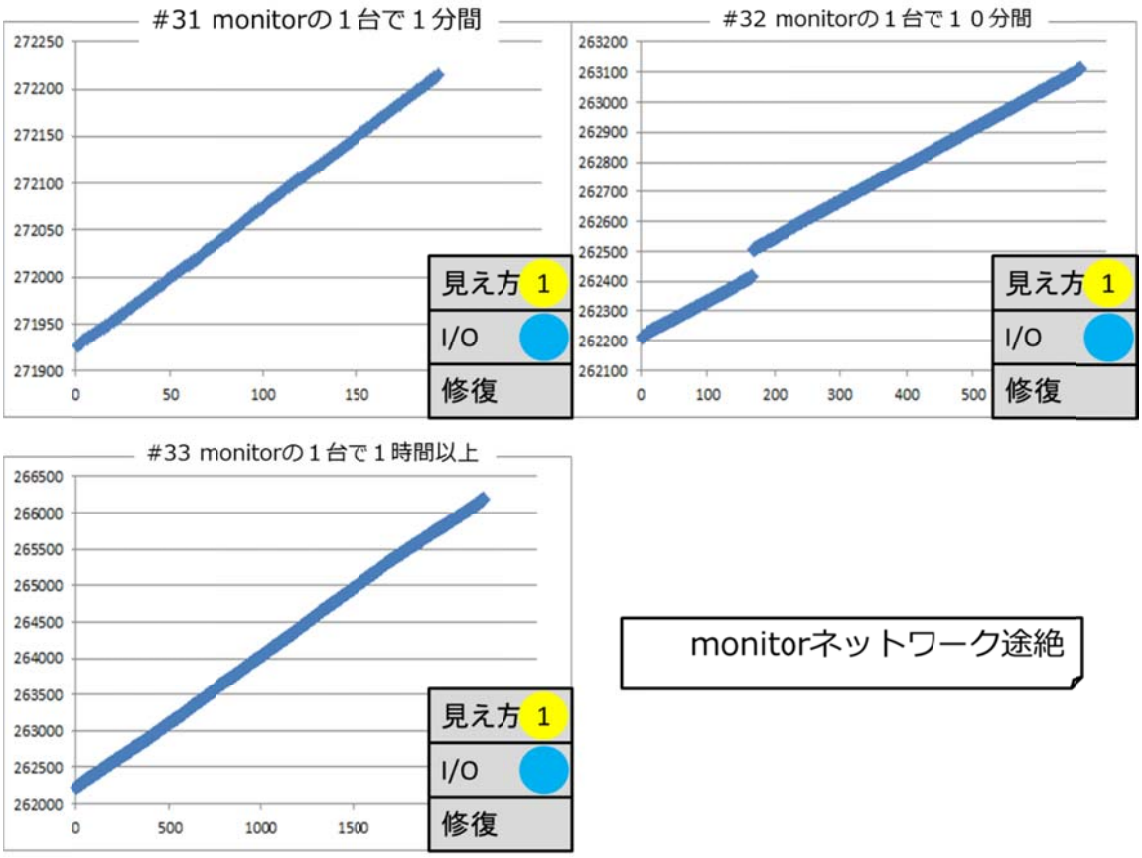
テスト項目 #2 8 については当該 Ceph OSD を一旦削除した後、故障し た NIC を交換し、新規 Ceph OSD として再度追加することでも復旧可能です。



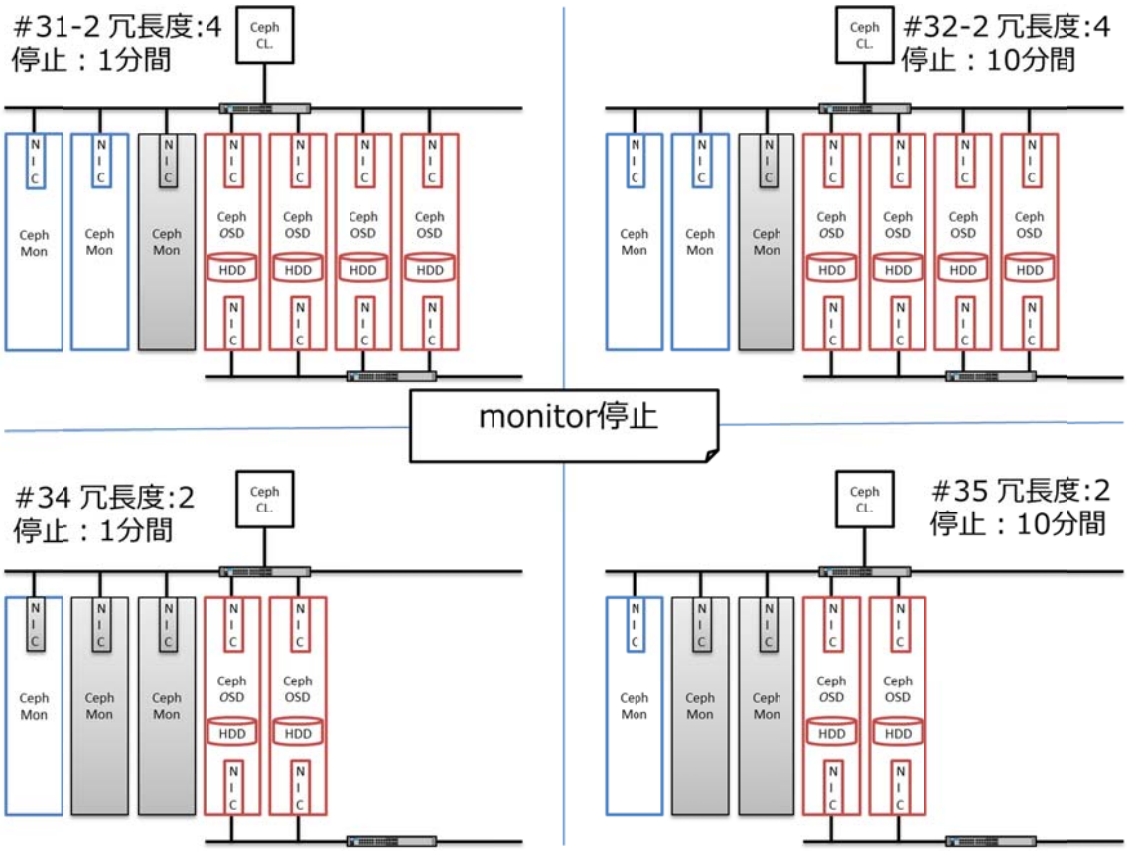
テスト項目 #31 から #33 です。

ここでは Ceph MON サーバーの 1 台でネットワーク途絶が発生しています。各サーバーは、public network のネットワークから途絶しています。cluster network のネットワークには接続されていません。ネットワーク途絶の状態が持続した時間が余白に記載されています。

Ceph は中央のサーバーがなく、各 Ceph クライアントが Ceph OSD とオブジェクトを直接にやりとりします。また、各 Ceph OSD はオブジェクトの複製を別のノード上に作成し、高可用性を確保します。このときの各 Ceph OSD 間も直接にやりとりします。Ceph クライアントと Ceph OSD はオブジェクトの所在を計算するのに CRUSH アルゴリズムを使用します。CRUSH アルゴリズムは、各 Ceph クライアント、各 Ceph OSD が Ceph ストレージ・クラスタのトポロジーを知っていることに依存します。Ceph ストレージ・クラスタのトポロジーはクラスタマップで管理されます。Ceph MON はクラスタマップを管理します。Ceph クライアント、Ceph OSD はクラスタマップのコピーを持ち、Ceph MON に問い合わせることなくデータにアクセスすることができます。

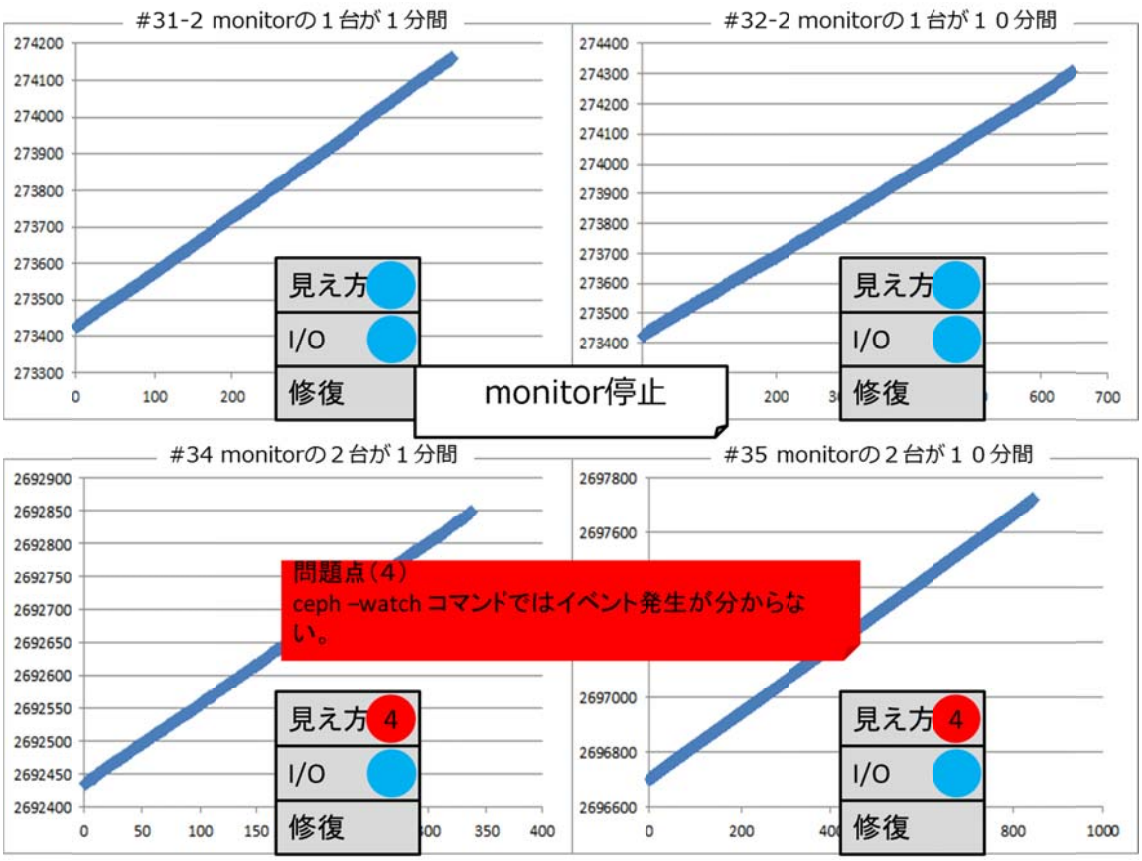


テスト項目 #31 から #33 では Ceph MON の 1 台でネットワーク途絶が発生している間も I/O が継続されていたことが分かります。ただし、テスト項目 #32 では、複数の RBD クライアントのうちの 1 台で約 90 秒間 I/O がブロックされていました。この時の Ceph ログには、4 台の Ceph OSD のうちの 3 台が、残り 1 台に対する監視の通信の応答を待っているログが出力されていました。Ceph MON がクラスタマップを最新に維持するため、各 Ceph OSD はランダムな Ceph MON1 台に対して定期的に通信を行いますが、当該 Ceph MON でネットワーク途絶が発生した影響でこの通信が滞り、連鎖的に、当該 Ceph OSD に対するその他の Ceph OSD からの監視の通信の応答に時間が掛かっていた可能性が考えられます。Ceph クライアントからのオブジェクト書き出しの I/O 要求は、CRUSH アルゴリズムで決まるプライマリ OSD が受信し、プライマリ OSD が CRUSH アルゴリズムで決まるレプリカ OSD に I/O 要求を配布し、レプリカ OSD からの応答を回収してから Ceph クライアントに応答します。よって、プライマリ OSD かレプリカ OSD のいずれかで通信の応答に時間が掛かる状況では、 Ceph クライアントの I/O 要求の応答にも時間が掛かるものと思われます。



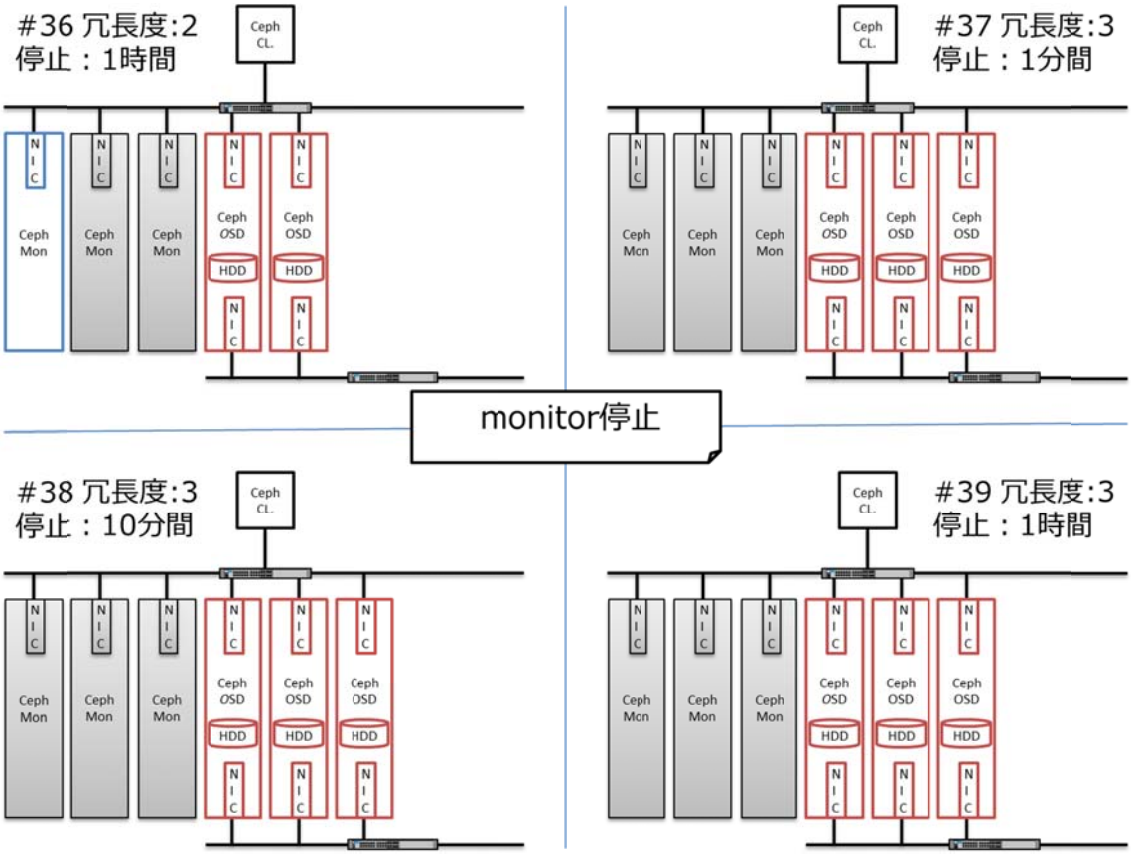
テスト項目# 31-2 から #35 です。

ここでは Ceph MON サーバーの 1 台または 2 台でダウンが発生しています。Ceph MON サーバーがダウンした状態が持続した時間が余白に記載されています。

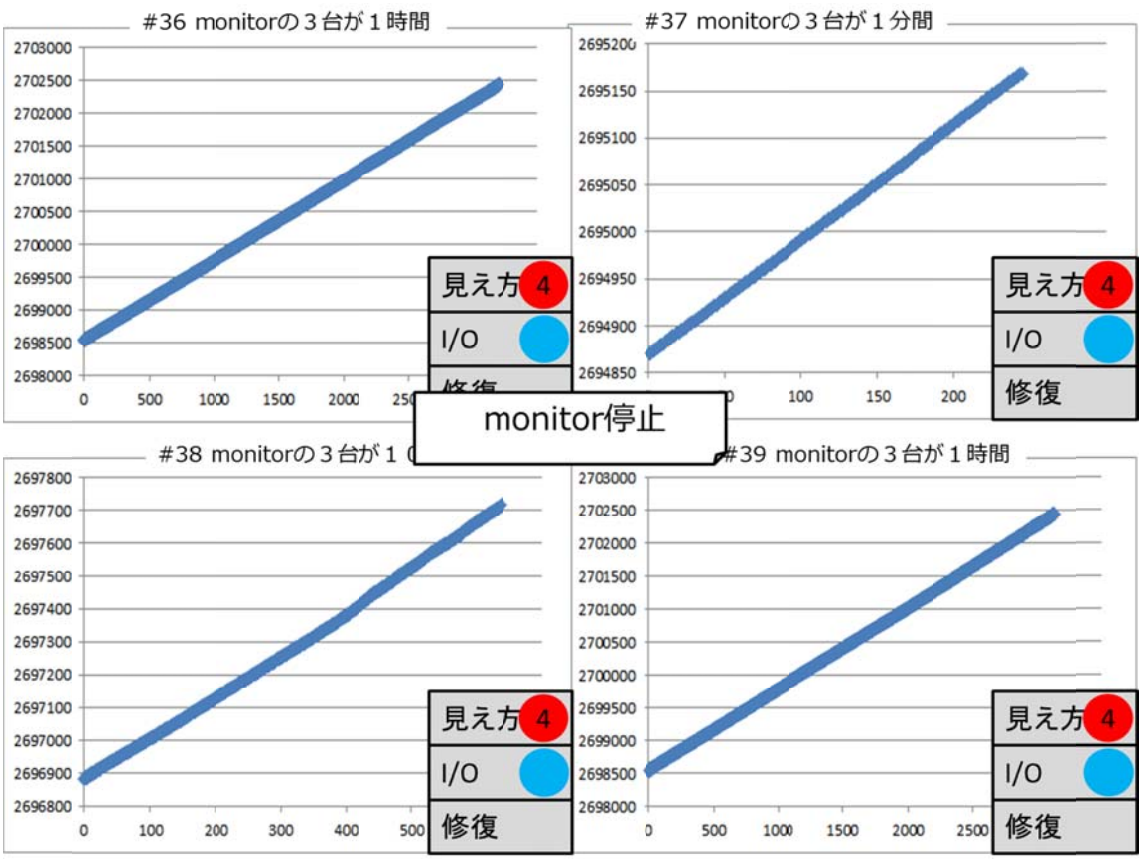


テスト項目 #32-2 から #35 では Ceph MON の 1 台または 2 台でダウンが発生している間も I/O が継続されていたことが分かります。ただし、テスト項目 #34 と #35 では、コマンド (ceph –watch) の出力から Ceph MON がダウンしていることが判別できませんでした。具体的には、2 台の Ceph MON がダウンしたにも関わらず、コマンド (ceph –watch) の出力上では、1 台の Ceph MON が停止し HELTH_WARN に状態遷移し、当該 Ceph MON 再起動後に HEALTH_OK に状態したように見え、もう 1 台の Ceph MON が停止したことが分かりませんでした。

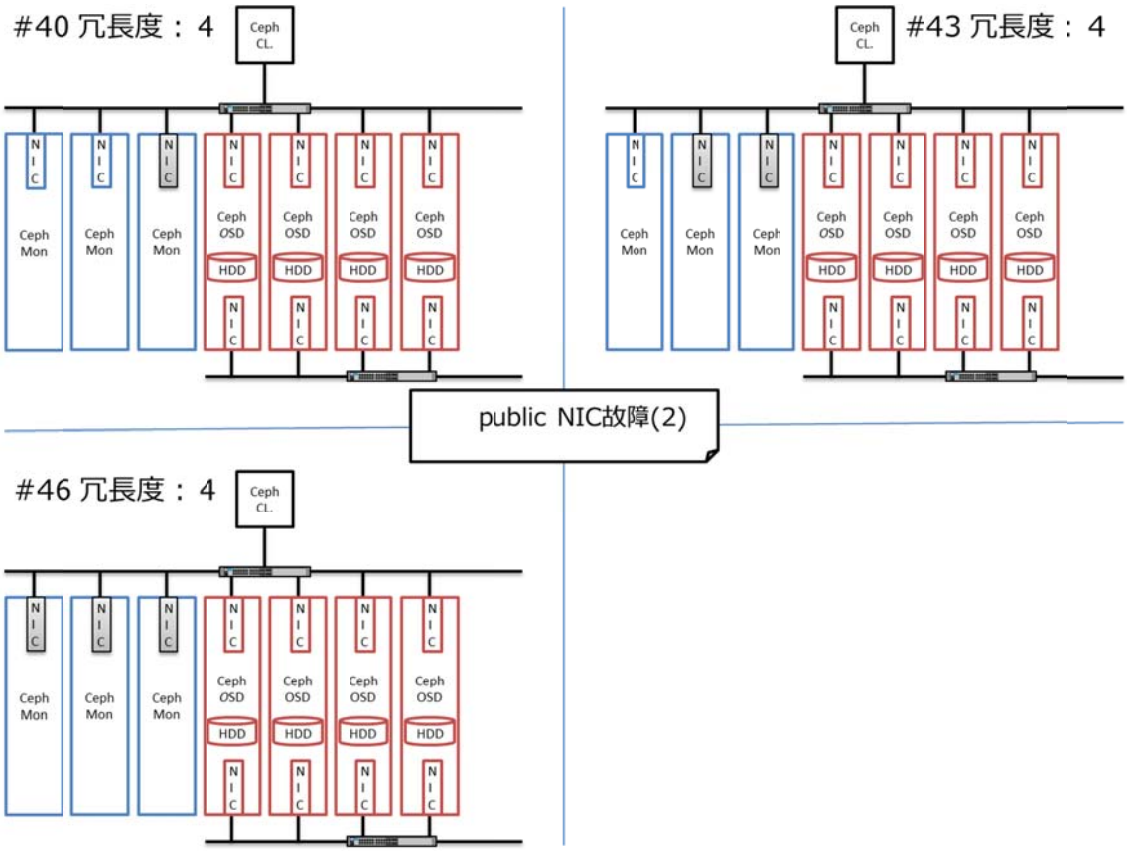
Ceph MON クラスタは奇数台数で構成され、その動作には多数派を形成できる台数 (＝定足数) が健全である必要があります。テスト項目 #34 と #35 では、3 台で構成された Ceph MON クラスタのうち 2 台がダウンしてしまうため、健全な Ceph MON の台数＝1 が定足数＝2 に満たない状況となります。また、コマンド (ceph –watch) の動作は Ceph MON クラスタが動作していることに依存します。よって、Ceph MON クラスタの定足数がない状態で発生した、もう 1 台の Ceph MON が停止、再起動の事象を、コマンド (ceph –watch) が正常に動作せず検知できなかったものと思われます。



Ceph MON サーバーダウンの項目がつづきます。テスト項目 #36 から #39 です。
ここでは Ceph MON サーバーの 2 台または 3 台全てでダウンが発生しています。いずれのテスト項目も Ceph MON クラスタが定足数に満たなくなり動作しなくなる状況です。

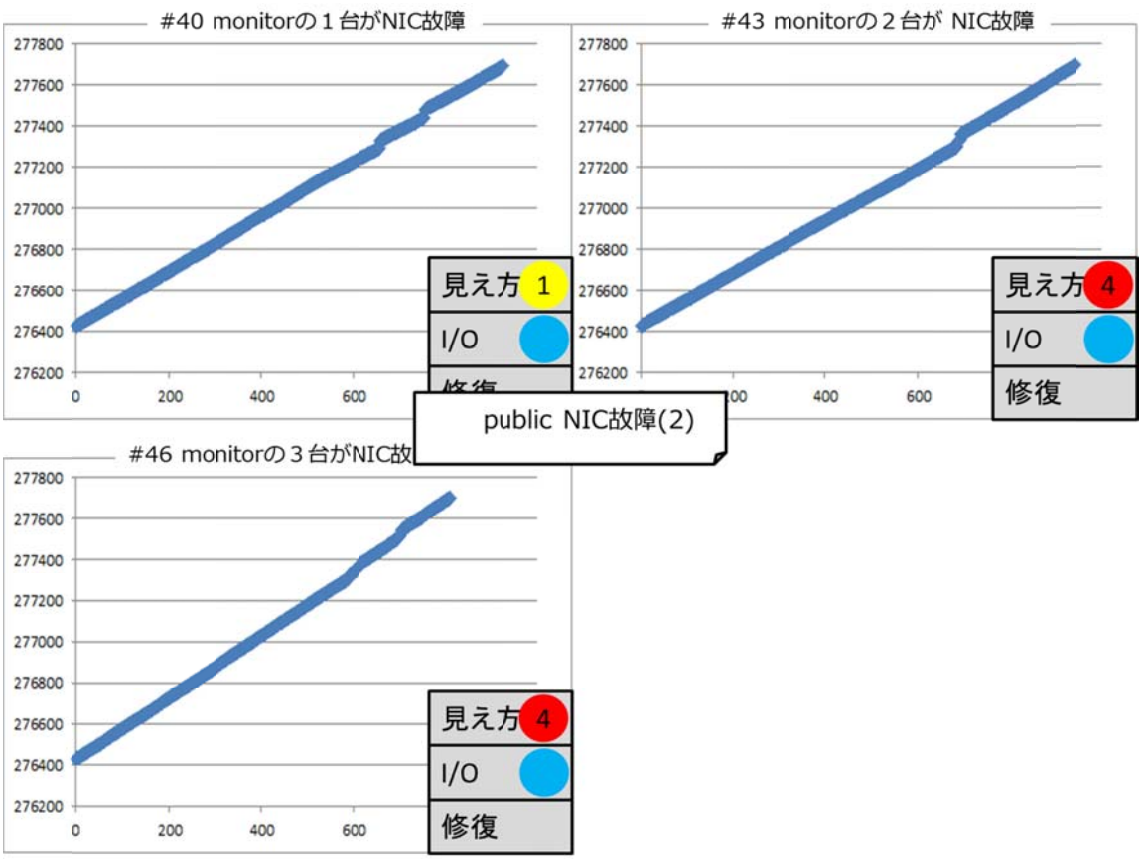


テスト項目 #32-2 から #35 と同様に #36 から #39 についても Ceph MON の 2 台または 3 台全てでダウンが発生している間も I/O が継続されていたことが分かります。いずれのテスト項目も、Ceph MON クラスタが定足数に満たない状況のため、コマンド (ceph –watch) の出力から Ceph MON がダウンしていることが判別できません。



テスト項目 #40 から #46 です。

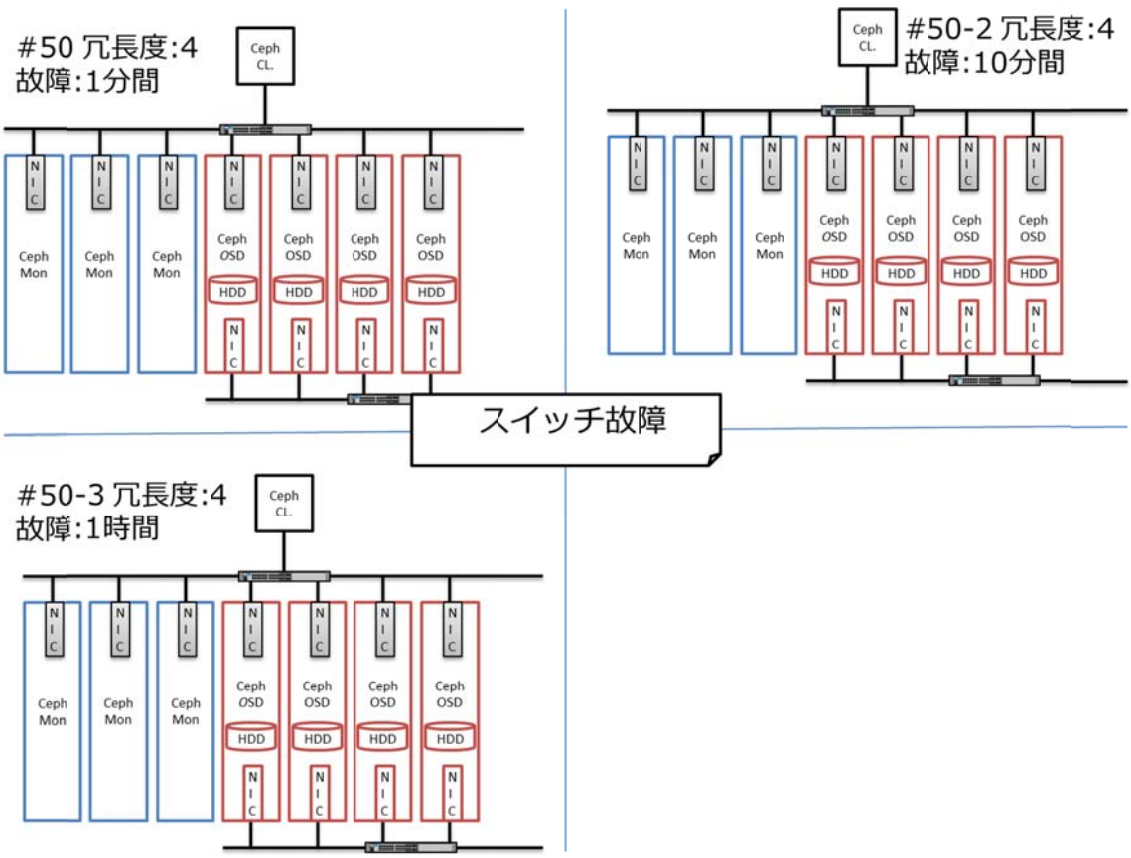
ここでは Ceph MON サーバーの 1 台から 3 台全てで NIC 故障が発生しています。各サーバーは、public network のネットワークにのみ接続されており、cluster network のネットワークには接続されていません。テスト項目 #43 と #44 は Ceph MON クラスタが定足数に満たなくなり動作しなくなる状況です。



Ceph MON がダウンする項目と同様に、Ceph MON の 2 台または 3 台全てで NIC 故障が発生している間も I/O が継続されていたことが分かります。テスト項目 #40 はコマンド (ceph –watch) 出力上、Ceph MON の NIC 故障は Ceph MON のダウンとして見えています。また、テスト項目 #43 と #46 は Ceph MON クラスタが定足数に満たない状況のため、コマンド (ceph –watch) の出力から Ceph MON の NIC が故障していることが判別できません。

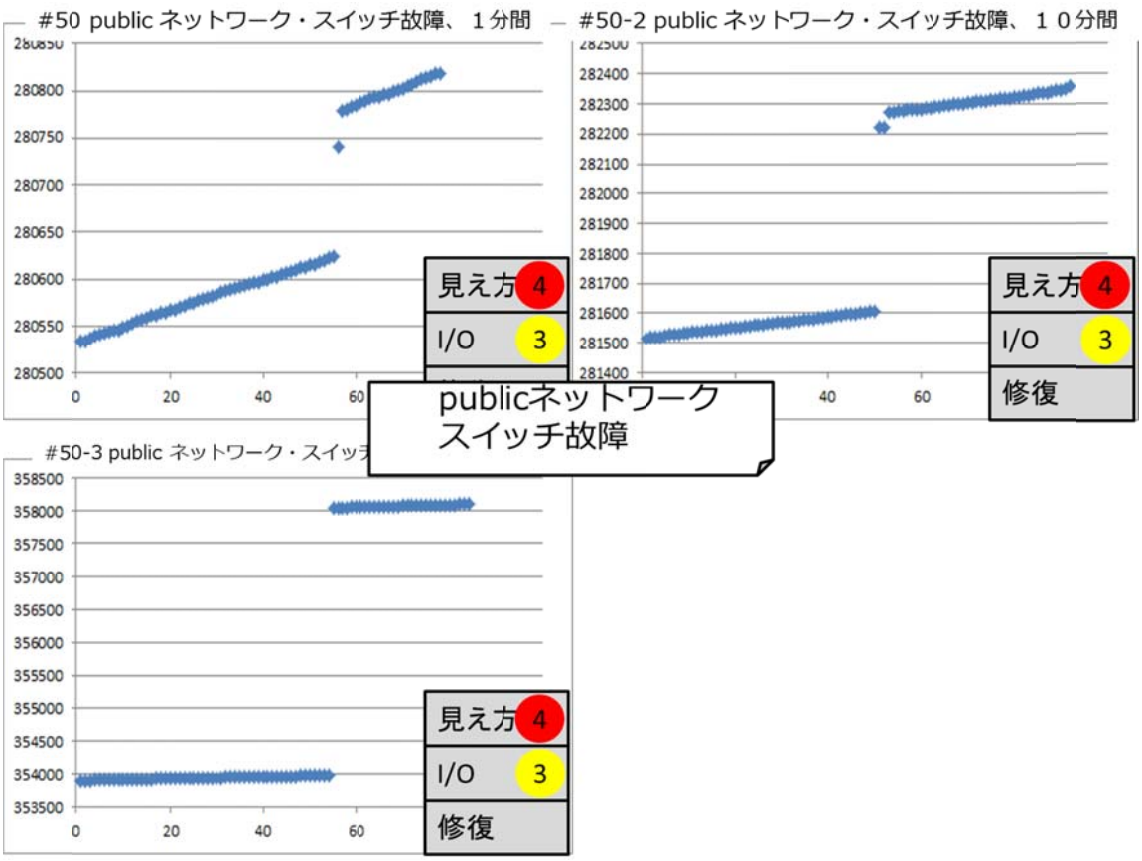
Ceph MON の NIC 故障の復旧方法としては Ceph MON を交換する (当該 Ceph MON を削除し、新規 Ceph MON を追加する) 方法と、故障した NIC のみ交換する (当該 Ceph MON をシャットダウンし、NIC を交換後再起動する) 方法がありますが、前者については Ceph MON を削除するオペレーションが Ceph MON クラスタが動作している、すなわち健全な Ceph MON 台数が定足数残っていることに依存する点に注意が必要です。

本稿では Ceph MON クラスタが動作しているテスト項目 #40 でのみ Ceph MON の交換による方法で対処し、Ceph MON クラスタが定足数に満たないテスト項目 #43 と #46 については NIC のみ交換する方法で対処しました。

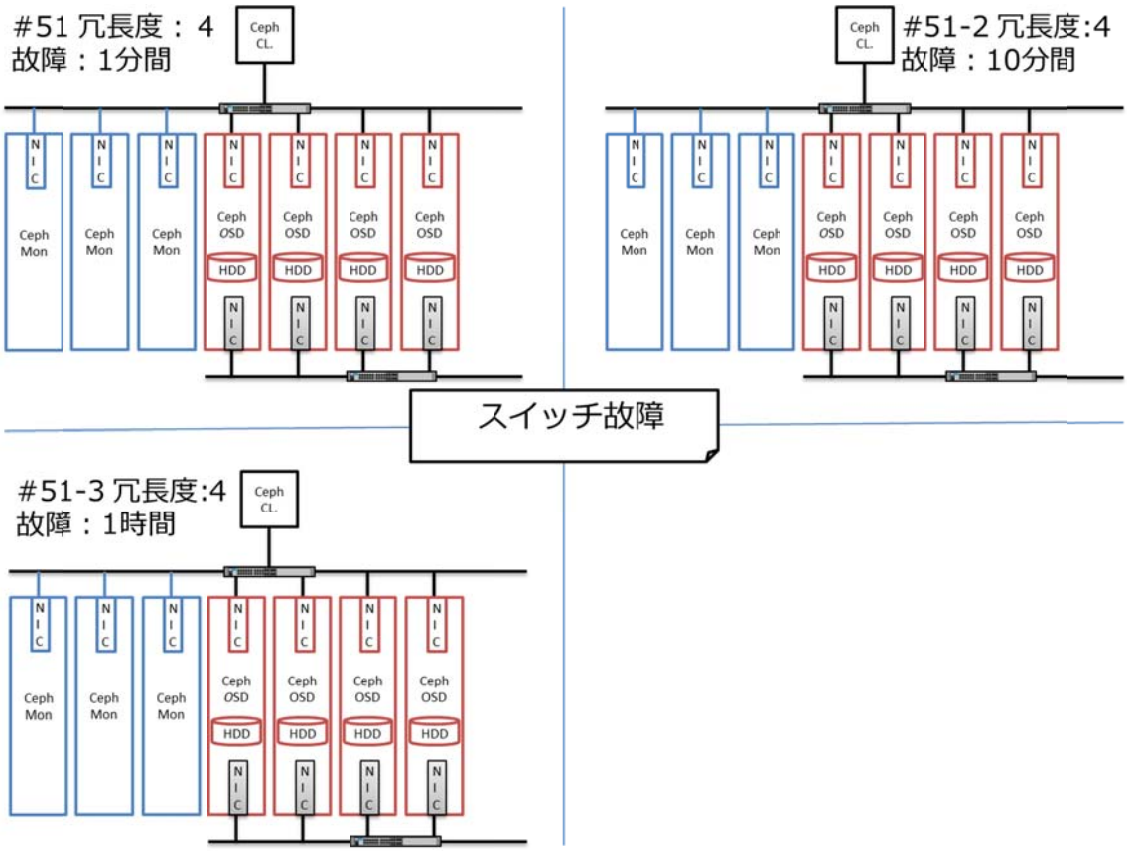


テスト項目 #50 から #50-3 です。

ここではネットワーク 2 系統のうち、public network のスイッチのダウンが発生しています。スイッチがダウンした状態が持続した時間が余白に記載されています。

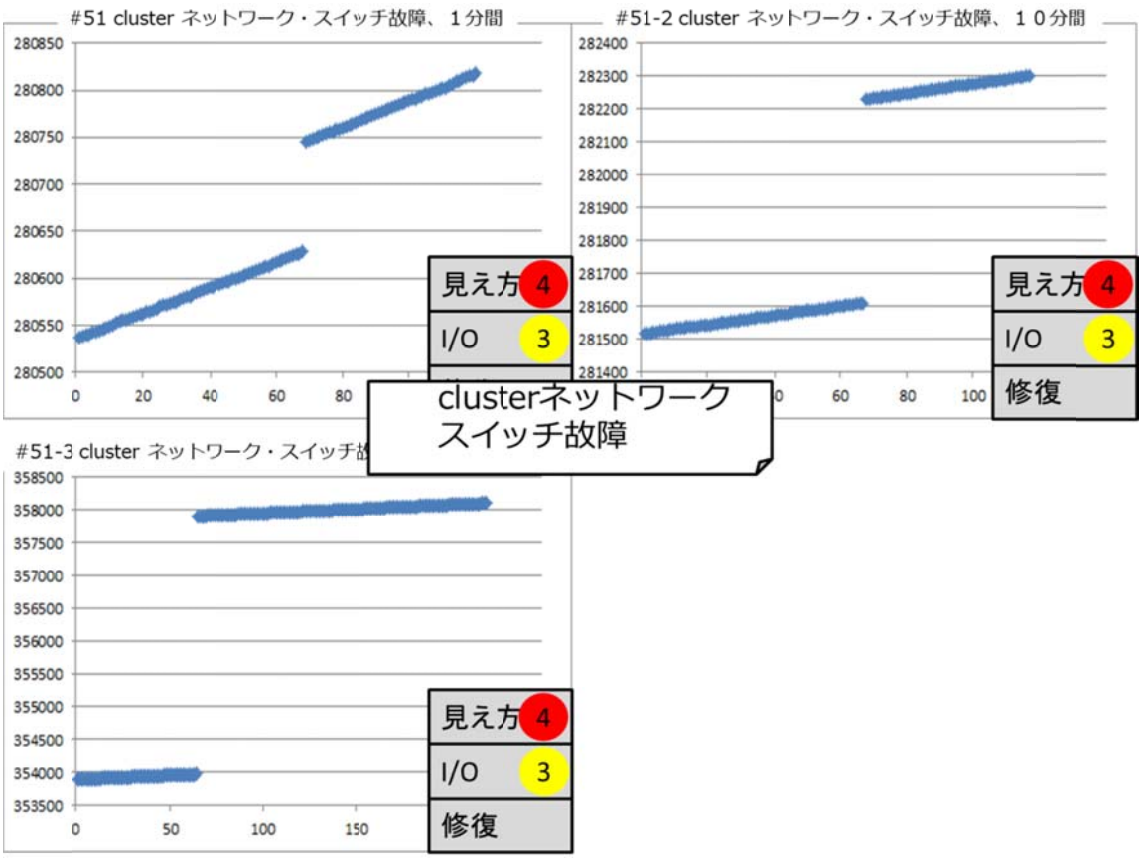


いずれのテスト項目も public network のスイッチがダウンしている間 I/O がブロックされていたことが分かります。I/O プロセスはいずれも終了していないので、public network のスイッチが復旧した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。public network のダウンはコマンド (ceph –watch) と Ceph MON クラスタの通信もできないため、コマンド (ceph –watch) 出力から public network がダウンしたことを判別することはできません。



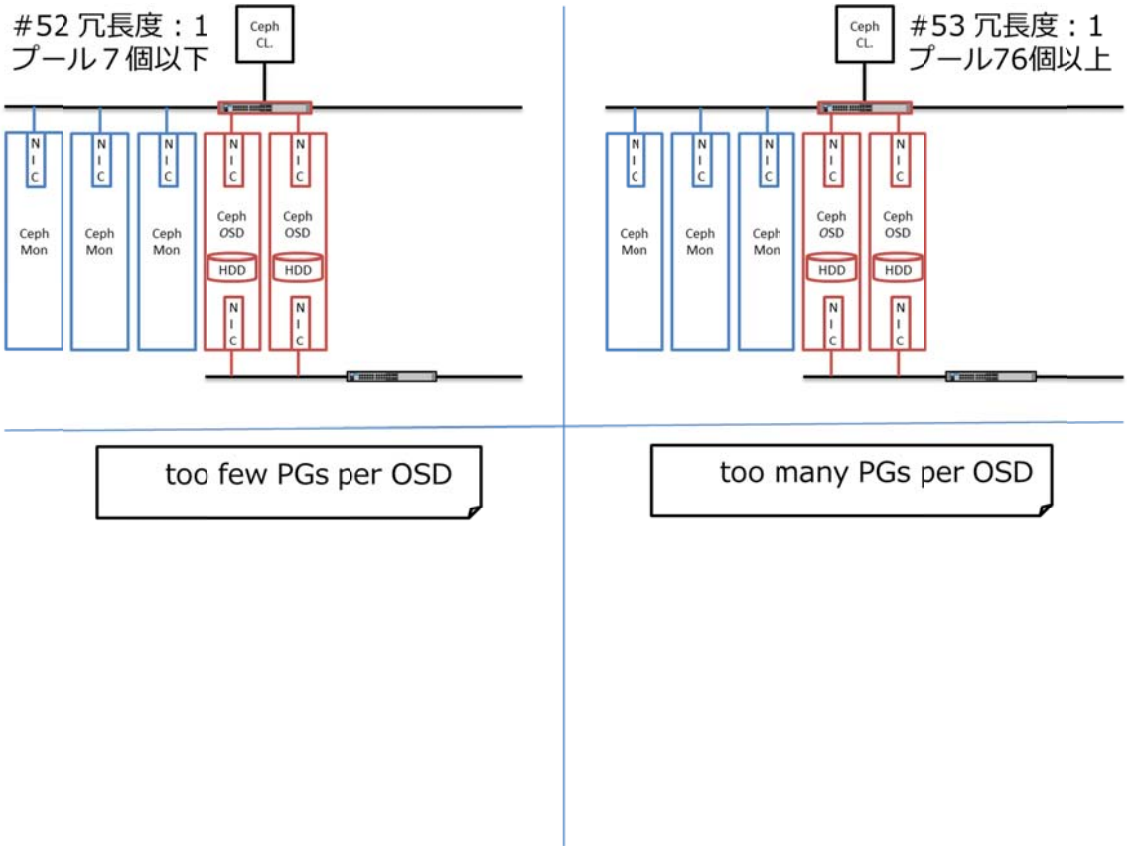
テスト項目 #51 から #51-3 です。

ここではネットワーク 2 系統のうち、cluster network のスイッチのダウンが発生しています。スイッチがダウンした状態が持続した時間が余白に記載されています。



いずれのテスト項目も cluster network のスイッチがダウンしている間 I/O がブロックされていたことが分かります。I/O プロセスはいずれも終了していないので、cluster network のスイッチが復旧した時点で I/O のブロックが解け、I/O エラーは発生していないことが分かります。cluster network は、Ceph OSD 間の通信に使用されます。Ceph クライアントからのオブジェクト書き出しの I/O 要求は、CRUSH アルゴリズムで決まるプライマリ OSD が受信し、プライマリ OSD が CRUSH アルゴリズムで決まるレプリカ OSD に I/O 要求を配布し、レプリカ OSD からの応答を回収してから Ceph クライアントに応答します。よって、プライマリ OSD とレプリカ OSD の間の通信ができない状況のため、Ceph クライアントの I/O 要求の応答がなかったものと思われます。

いずれのテスト項目も、コマンド (ceph –watch) の出力から cluster network のスイッチダウンを判別することはできませんでした。具体的には、各 Ceph OSD がダウンとダウンからの復旧を繰り返しているように見えていました。これは、cluster network がダウンしたことで、Ceph OSD 間の監視の通信ができなくなったことで、Ceph OSD 同士が互いを Ceph OSD ダウンとしたと判定しこれを Ceph MON に報告する一方で、public network はダウンしていないので、各 Ceph OSD はランダムな Ceph MON 1 台に対して定期的な通信を継続するため、Ceph MON がこれをダウンした Ceph MON の復旧あるいは先のダウン報告の誤報と判定しているものと思われます。

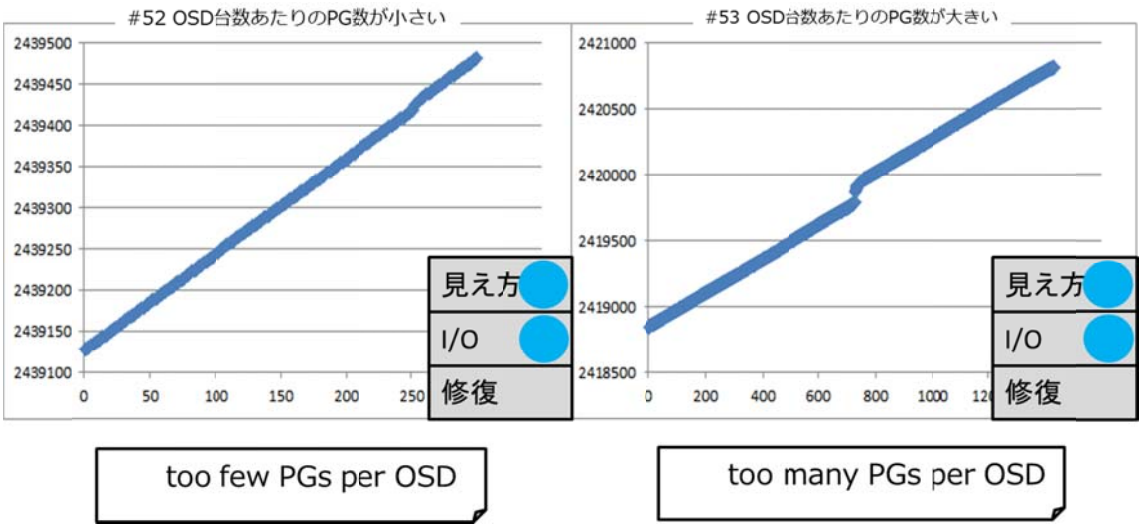


最後の項目は Ceph OSD 台数あたりの PG 数が少なすぎる、あるいは多すぎる場合に関する項目です。テスト項目 #52 は PG 数が少なすぎる場合で、テスト項目 #53 は PG 数が多すぎる場合です。

本稿の構成では FileStore と呼ばれるストレージバックエンドが使用されているので、オブジェクトは Ceph OSD 上のファイルです。可用性のため、オブジェクトのレプリカが複数の Ceph OSD 上に置かれます。あるオブジェクトのレプリカを置く Ceph OSD セットのパターンを Placement Group (PG) といいます。PG はプールを作成すると Ceph OSD 数に応じた数が自動的に定義されます。オブジェクトと Ceph OSD のマッピングは直接的ではなく間接的に行われます。オブジェクトは一旦 PG にマッピングされ、PG が Ceph OSD にマッピングされます。プールは、1 つの Ceph ストレージ・クラスタを複数の用途で使用する場合の、用途毎の論理的なパーティションで、データ冗長度等はプール毎に設定する属性です。よって、プールの数は運用設計に依存します。

PG 数の調節はプール数とプール当たりの PG 数のパラメータで可能です。オブジェクトの I/O およびデータ冗長度の維持は PG 毎に行われるので、Ceph OSD 数に応じた PG 数を設定します。Ceph OSD 数に対して現在の PG 数が多すぎたり少なすぎたりすることは性能上好ましくないため、いずれの場合も Ceph クラスタの状態が HEALTH_OK から HEALTH_WARN に遷移し、ワーニングメッセージが出力されます。PG 数は予め設計・設定するものなので、通常の運用においてこれらのワーニングが出力される状況はプール数あるいは Ceph OSD 数の増減があった状況等が考えられます。

本稿では性能上の観点は設定していないので、いずれのテスト項目においても I/O が継続されていることのみを確認します。



いずれのテスト項目もワーニングメッセージが出力されている間も I/O が継続されていたことが分かります。テスト項目 #53 で I/O の進捗が滞っている箇所が見られますが、これは Ceph OSD の 1 台で意図しないディスク故障が発生し、Ceph OSD デーモンが EIO エラーにより再起動を繰り返していたためで、本テスト項目との関係はありません。当該 Ceph OSD の対処を行うことで現象が収束することを確認しています。

OpenStack/Ceph 異常系テスト (1) の具体的なテスト項目 (想定されるハードウェア障害) とその結果は以上です。

今回設定したシステムの構成とハードウェア障害のパターンの範囲において、アプリケーションの I/O への影響という観点に関しては、総じて非常に分かりやすい結果が出たと思われます。すなわち、ハードウェアの故障によってアプリケーションの I/O がブロックされる場合があるが、ハードウェアの故障を取り除くことによって I/O のブロックは解ける、という見え方です。
一方で運用上の問題も検出されました。すなわち、コマンド (ceph -watch) 出力でハードウェア障害のイベント発生が分からないことや、コマンド (ceph -watch) 出力と実際のイベントが一致しないことがあるという点です。
フィールドサポートの場面としては障害箇所の特定のためのオペレーション (コマンド実行等) をタイムリーに実施することが困難な場合も想定されます。このような場合に対しては取得済みの Ceph のログからの障害箇所の特定が必要と思われます。
上記問題のフィードバックとして、この後に述べる OpenStack/Ceph 異常系テスト (2) においては、対処方法の情報として、ログだけからハード障害を切り分けできるかどうかという観点を加えています。そのために、テスト (2) の実施に先立ってまず現状の Ceph ログの問題点と対処案を検討します。

4. Ceph のログ使い勝手の改善

OpenStack/Ceph 異常系テスト (1) では、特定のシステム構成で発生するハードウェア障害のパターン毎に、OpenStack インスタンス (VM) 上のアプリケーション I/O への影響と、コマンド (ceph –watch) 出力等に基づいた障害切り分けから復旧までの手順を見てきました。アプリケーション I/O への影響については、ハードウェアの故障を取り除くまでアプリケーションの I/O がブロックされる場合があることが分かりました。障害切り分けから復旧までの手順について、コマンド (ceph –watch) 出力だけでは障害箇所の特定ができない場合があることが分かりました。
そこで、ここからはログから障害の切り分けを行うことを想定した場合の、現状の Ceph ログの問題点とその対処案を検討します。

4.1. Ceph ログの問題点

まず、筆者の主観で Ceph ログの問題点を提示します。

対象の Ceph バージョンは Infernalis (v9.2.0) です。
本稿執筆時点 (2017 年 8 月末) で Ceph バージョンは Luminous (v12.2.0) です。なお、10 月 Web サイト掲載時点での最新バージョンは、9 月 28 日にリリースされた Luminous (v12.2.1) です。

以下に述べる問題点の Jewel (v10.2.0) バージョン以降での状況は未確認です。

主に 2 つの難しい点があると考えています。1 つはログレベルの設定で、もう一つはログの解析です。

<ログレベルの設定の難しさ>

Subsystem	Log Level	Memory Level
default	0	5
lockdep	0	5
context	0	5
crush	1	5
mds	1	5
mds balancer	1	5
mds locker	1	5
mds log	1	5
mds log expire	1	5
mds migrator	1	5
buffer	0	0
timer	0	5
filer	0	5
objecter	0	0
rados	0	5
rbd	0	5
journaler	0	5
objectcacher	0	5
client	0	5
osd	0	5
optracker	0	5
objclass	0	5
filestore	1	5
journal	1	5
ms	0	5
mon	1	5
monc	0	5
paxos	0	5
tp	0	5
auth	1	5
finisher	1	5
heartbeatmap	1	5
perfcounter	1	5
rgw	1	5
javaclient	1	5
asok	1	5
throttle	1	5

上記の表は、ログレベル設定に用いるパラメータの一覧とそれらのデフォルトの設定値です。サブシステム毎にログレベルとメモリログレベルの値を設定します。ログレベルというのは、ログファイルに出力するログ（以降、ファイルログと呼びます）のレベルです。メモリログレベルというのは、Ceph MON デーモンや Ceph OSD デーモン等のメモリ上のログ領域にサイクリックに記録するログ（以降、メモリログと呼びます）のレベルです。ある時点のメモリログはコマンド（ceph log dump）を実行することでファイルログにダンプすることができます。

ログレベルの値は高いほどログを出力するソースコード上のログ出力箇所が多くなるため、ログ出力頻度が上がります。ログレベルの値として有効な範囲は -1 から 40 です。

ファイルログレベル	ログ行数/秒	ログサイズ(KB)/秒
5	47	4.9
10	80	10.2
15	82	10.6
20	121	16.3
25	127	17.6
30	139	18.8

上記の表は、6 台のサーバー (Ceph MON x 3 台と Ceph OSD x 3 台) で構成された Ceph クラスタで、I/O 中のアプリケーションのないアイドル状態で出力されたログの量から求めたおよそのログ出力頻度です。全てのサブシステムについて同じログレベルを設定しています。

この結果からは、比較的低いログレベルの設定でもログ出力頻度は運用への影響が無視できないレベルであることが分かります。ログ出力による運用への影響は、ログの量だけでなく、I/O 性能の顕著な劣化としても現れる場合があります。また、メモリログに関してはメモリ上のログ領域にサイクリックに記録するので、メモリログレベルを高く設定すると、より広範囲のログが取得できる可能性がある反面、過去のログが上書きされるまでの時間が短くなり、障害発生時にメモリログをダンプするのに十分な時間的猶予を確保できなくなる可能性があります。よって、運用への影響を回避しつつ、できるだけ詳細なログをできるだけ長時間確保するための、サブシステム毎の最適なログレベル/メモリログレベル設定が求められます。

	-1	0	1	2	3	4	5	6	7	10	11	12	14	15	20	21	25	30	35	40
ceph_subsys_	10	3	1							1										
ceph_subsys_asok		2					15								3			4		
ceph_subsys_auth	3	34		1			1			55					6			8		
ceph_subsys_civetweb		1	1																	
ceph_subsys_client	5	50	60	36	112	4	29	1	2	236		5		32	65	1	7	2		
ceph_subsys_compressor			2							10					4					
ceph_subsys_crush		2	11	9			28			2					1					
ceph_subsys_crypto		15													9					
ceph_subsys_filer										10										
ceph_subsys_filestore	105	123	14	5			16			178				48	60		10	2		
ceph_subsys_finisher										11										
ceph_subsys_heartbeatmap		3	2							2					3					
ceph_subsys_javaclient										109					1					
ceph_subsys_journal	37	10	8	20	5		5			59				4	27		2			
ceph_subsys_journaler	1	5	14			2				37		1			3					
ceph_subsys_keyvaluestore	27	9	7	1			9			80				30	17			24		
ceph_subsys_mds	138	92	85	16	14	113	66	3	432	1360	1	32	5	112	218		6	1	2	
ceph_subsys_mon	49	67	61	10	1	8	53		37	393	1	2		15	61		3	8		
ceph_subsys_monc		5	3				2			59					4					
ceph_subsys_ms		83	86	26	3		7			217				6	104		5	4		
ceph_subsys_newstore	38	2	10	3			2			72				31	85			45		1
ceph_subsys_objectcacher			1						8	88	1				23			2		
ceph_subsys_objecter		3	1	1	8	1	6		8	109				18	34					
ceph_subsys_optracker							1			1										
ceph_subsys_osd	126	52	22	6	4	2	26		47	923		7		36	324		19	29		
ceph_subsys_paxos	1		5				6		4	90				1	8			8		
ceph_subsys_rados		2	6							37					17					
ceph_subsys_rbd			9	30			62			58				6	146					
ceph_subsys_refs			2																	
ceph_subsys_rgw	20	322	14	19			112			113				19	168					
ceph_subsys_striper										5				1	13			4		
ceph_subsys_throttle				1	1					6										
ceph_subsys_timer										12					2					
ceph_subsys_tp			2							25		1		6	1					
ceph_subsys_xio		5		13		22	1			3	1				1		4			

上記の表は、サブシステムおよびログレベル毎の、ソースコード上のログ出力箇所の数をカウントした結果です。全部で約 9,700 箇所あります。紫色のマークはデフォルトのログレベル範囲を表します。つまり、ファイ

ルに残るログを出力している箇所です。水色のマークはデフォルトでメモリログレベル範囲を表します。つまり、メモリ上にサイクリックに記録されるログを出力している箇所です。
サブシステム毎の最適なログレベル/メモリログレベル設定には、これらのログ出力箇所のコードが実行される頻度に関する情報が必要と思われます。

	-1	0	1	2	3	4	5	6	7	10	11	12	14	15	20	21	25	30	35	40
ceph_subsys_																				
ceph_subsys_asok							6											24		
ceph_subsys_auth										943					396			54		
ceph_subsys_civetweb																				
ceph_subsys_client																				
ceph_subsys_compressor																				
ceph_subsys_crush																				
ceph_subsys_crypto																				
ceph_subsys_filer																				
ceph_subsys_filestore							370			1078				240	357					
ceph_subsys_finisher										573										
ceph_subsys_heartbeatmap															1701					
ceph_subsys_javaclient																				
ceph_subsys_journal							107			850				65	356					
ceph_subsys_journaler																				
ceph_subsys_keyvaluestore														56						
ceph_subsys_mds										342										
ceph_subsys_mon		18	26	54					55	933	25			70	1908		6	25		
ceph_subsys_monc										72										
ceph_subsys_ms			1220	10						5792				681	8943			2478		
ceph_subsys_newstore							65													
ceph_subsys_objectcacher																				
ceph_subsys_objecter																				
ceph_subsys_optracker							5045			1										
ceph_subsys_osd							81		89	1881				371	656		193	781		
ceph_subsys_paxos							172		25	1236					131					
ceph_subsys_rados																				
ceph_subsys_rbd										44					217					
ceph_subsys_refs			14672																	
ceph_subsys_rgw																				
ceph_subsys_striper																				
ceph_subsys_throttle										2369										
ceph_subsys_timer										493					280					
ceph_subsys_tp										12		108		120	170					
ceph_subsys_xio																				

上記の表は、ある検証項目を実施したときに 6 ノードの Ceph クラスタで出力されたログのログレベル分布です。 セルの値はログ出力件数です。頻度の高いものは赤、比較的低いものは緑色になっています。各 Ceph ノードのメモリ上のログ領域は 1 万行に設定されているため、ログは全部で 6 万件あります。 しかし、このログでカバーしている時間的範囲は 6 ノード合計で 210 秒程度に過ぎません。 つまり、問題が発生からログを取得するまでの時間的猶予が非常に限られている状況です。現実的な時間的猶予でログを取得できるようにするためにはメモリ上のログ領域を拡大するだけでなくセルの赤い箇所のログを削減する必要があると思われます。

	-1	0	1	2	3	4	5	6	7	10	11	12	14	15	20	21	25	30	35	40
ceph_subsys_																				
ceph_subsys_asok							12											24		
ceph_subsys_auth										1416					458			120		
ceph_subsys_civetweb																				
ceph_subsys_client																				
ceph_subsys_compressor																				
ceph_subsys_crush																				
ceph_subsys_crypto																				
ceph_subsys_filer																				
ceph_subsys_filestore							300			853				350	352					
ceph_subsys_finisher										947										
ceph_subsys_heartbeatmap																				
ceph_subsys_javaclient																				
ceph_subsys_journal							158			954				75	792					
ceph_subsys_journaler																				
ceph_subsys_keyvaluestore														141						
ceph_subsys_mds										389										
ceph_subsys_mon		20	14	93					56	1025	33			60	2192		7	75		
ceph_subsys_monc										139										
ceph_subsys_ms			2397	37																
ceph_subsys_newstore							75			51										
ceph_subsys_objectcacher																				
ceph_subsys_objecter																				
ceph_subsys_optracker							5796													
ceph_subsys_osd			19				374		108	3219				459	1323		696	1989		
ceph_subsys_paxos							211		40	1191					139					
ceph_subsys_rados																				
ceph_subsys_rbd										25					228					
ceph_subsys_refs																				
ceph_subsys_rgw															2					
ceph_subsys_striper																				
ceph_subsys_throttle																				
ceph_subsys_timer										1208					952					
ceph_subsys_tp										64		75		107	739					
ceph_subsys_xio																				

上記の表は、先程の表でセルが赤かった箇所のログを出ないようにログレベルを変更した後の様子です。先程と同じく 6 ノード全部で 6 万行のログ領域に対して 6 ノード合計で 330 秒間のログを格納してログは全部で 3 万件に抑えられています。つまり、検証項目を実施したときに 6 ノードの Ceph クラスタで出力されたログが全て残っている状況です。

このようにログ出力頻度の高い箇所を避けるようにログレベルを設定することでメモリ上にログが残る時間を延長することが可能です。本稿での検証は、90 分間のメモリログが取得できるログレベル設定を目標とします。

4.1. Ceph ログの問題点 (つづき)

<ログの解析の難しさ>

```
7fe4d6eb6700 20 timer(0x7fe4e6cb6070).timer_thread awake
7fe4d6eb6700 10 timer(0x7fe4e6cb6070).timer_thread executing 0x7fe4ebd6dfb0
7fe4d6eb6700  5 osd.1 45634 tick
7fe4d6eb6700 25 osd.1 45634 heartbeat_check osd.0 first_tx 2016-08-23 04:04:28.573642 last_tx 2016-08-24 06:56:04.650271 last_rx_back 2016-08-24 06:56:08.650271
last_rx_front 2016-08-24 06:56:08.650271
7fe4d6eb6700 25 osd.1 45634 heartbeat_check osd.2 first_tx 2016-08-24 05:54:57.132631 last_tx 2016-08-24 06:56:04.650271 last_rx_back 2016-08-24 06:56:08.650271
last_rx_front 2016-08-24 06:56:08.650271
7fe4d6eb6700 10 osd.1 45634 do_waiters -- start
7fe4d6eb6700 10 osd.1 45634 do_waiters -- finish
7fe4d6eb6700 10 timer(0x7fe4e6cb6070).add_event_at 2016-08-24 06:56:15.987674 -> 0x7fe4ebd6d8b0
7fe4d6eb6700 20 timer(0x7fe4e6cb6070).timer_thread going to sleep
7fe4bf30e700 20 OSD::recovery_tp worker waiting
7fe4bf30e700 20 heartbeat_map reset_timeout 'OSD::recovery_tp thread 0x7fe4bf30e700' grace 60 suicide 0
7fe4c631c700 20 timer(0x7fff20483048).timer_thread awake
7fe4c631c700 10 timer(0x7fff20483048).timer_thread executing 0x7fe4e6af1940
7fe4c631c700 10 monclient: tick
7fe4c631c700 10 cephx: validate_tickets want 37 have 37 need 0
7fe4c631c700 20 cephx client: need_tickets: want=37 need=0 have=37
7fe4c631c700 10 monclient: _check_auth_rotating have uptodate secrets (they expire after 2016-08-24 06:55:44.088749)
7fe4c631c700 10 auth: dump_rotating:
7fe4c631c700 10 auth:  id 9391 AQCXJr1X3kWTHxAA3Hg7X+PboCQ+SqBuKF3x+A== expires 2016-08-24 06:46:15.529726
7fe4c631c700 10 auth:  id 9393 AQCpNL1XWpq1OBAAfhCllzorh1kozoEXKkZM0w== expires 2016-08-24 07:46:17.951407
7fe4c631c700 10 auth:  id 9394 AQC4Qr1Xu+CcDRAA93U3jUm3shKT6x5y8k+eVg== expires 2016-08-24 08:46:17.951407
7fe4c631c700 10 monclient: renew subs? (now: 2016-08-24 06:56:14.088802; renew after: 2016-08-24 06:58:20.055689) -- no
7fe4c631c700  1 RefCountedObject::get 0x7fe4ebcf7000 2 -> 3
7fe4c631c700 20 -- 10.100.30.11:6800/14971 send_keepalive con 0x7fe4ebdfa3c0, have pipe.
7fe4c631c700  1 RefCountedObject::put 0x7fe4ebcf7000 3 -> 2
7fe4c631c700 10 timer(0x7fff20483048).add_event_at 2016-08-24 06:56:24.188830 -> 0x7fe4e6afd400
7fe4c631c700 20 timer(0x7fff20483048).timer_thread going to sleep
7fe4baa04700 10 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebcfa3c0).writer: state = open
policy.server=0
7fe4baa04700 10 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebcfa3c0).write_keepalive2 14
2016-08-24 06:56:14.088858
7fe4baa04700 10 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebcfa3c0).writer: state = open
policy.server=0
7fe4baa04700 20 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebcfa3c0).writer sleeping
7fe4b94f6700 20 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebcfa3c0).reader got
KEEPALIVE_ACK
7fe4b94f6700 20 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebcfa3c0).reader reading
tag...
7fe4c4b19700 20 OSD::osd_tp worker waiting
7fe4c4b19700 20 heartbeat_map reset_timeout 'OSD::osd_tp thread 0x7fe4c4b19700' grace 60 suicide 0
7fe4c531a700 20 OSD::osd_tp worker waiting
7fe4c531a700 20 heartbeat_map reset_timeout 'OSD::osd_tp thread 0x7fe4c531a700' grace 60 suicide 0
7fe4c0b11700 20 heartbeat_map reset_timeout 'OSD::osd_op_tp thread 0x7fe4c0b11700' grace 15 suicide 150
7fe4c0b11700 20 heartbeat_map reset_timeout 'OSD::osd_op_tp thread 0x7fe4c0b11700' grace 4 suicide 0
7fe4d24a2700 20 FileStore::op_tp worker waiting
7fe4d24a2700 20 heartbeat_map reset_timeout 'FileStore::op_tp thread 0x7fe4d24a2700' grace 60 suicide 0
7fe4d2ca3700 20 FileStore::op_tp worker waiting
7fe4d2ca3700 20 heartbeat_map reset_timeout 'FileStore::op_tp thread 0x7fe4d2ca3700' grace 60 suicide 0
7fe4da6b6700 20 timer(0x7fe4e6bbb980).timer_thread awake
7fe4da6b6700 10 timer(0x7fe4e6bbb980).timer_thread executing 0x7fe4e6af9f20
7fe4da6b6700 10 osd.1.objecter tick
7fe4da6b6700 10 timer(0x7fe4e6bbb980).add_event_at 2016-08-24 06:56:19.311697 -> 0x7fe4e6afd3d0
7fe4da6b6700 20 timer(0x7fe4e6bbb980).timer_thread going to sleep
```

上記は Ceph OSD サーバーで出力されたログのサンプルです。各ログ行は数カラムのタイムスタンプ部から始まりますが、その部分は省略しています。最初のカラムの 7fe4d6eb6700 のような値はスレッド毎の値です。2 番目のカラムはログレベルです。3 番目のカラム以降がログメッセージです。

障害発生時に上記ログから切り分けを行おうとした場合を想定すると、まず、各ログを出力しているソースコード上の場所を、ログメッセージに含まれるキーワード等でソースコードを検索して特定し、次に、前後のログ行を出力しているソースコード上の場所から処理の流れを類推することになります。

「1.2. Ceph を選択する課題」で述べたように、Ceph のコード規模は Linux カーネルに含まれるすべてのファイルシステムのコード (linux-4.7/fs) の総量におよそ匹敵する大きさがあります。現実的な時間的コストで、ログメッセージに含まれるキーワードでソースコードを検索してログ出力箇所を特定し、さらに複数のログ出力箇所からそれらを通る処理ルートを類推するためには、予め何らかの準備が必要と思われます。

4.2. Ceph ログの問題点への対処

ログから障害の切り分けを行うことを想定した場合の Ceph ログの問題点として、ログレベルの設定やログの解析の難しさについて述べました。 ここからは、これらの問題点に対して本稿で実施したアプローチについて説明します。

ログレベルの設定の難しさについては、以下の仮定に沿って試行しました。
仮定 1. 障害のパターン毎に実行される処理の集まりに傾向がある
仮定 2. 処理の集まりの傾向は、出力するログの傾向に現れる
仮定 3. 出力されたログの傾向から障害のパターンを類推できる

ログの解析の難しさについては、以下の仮定に沿って試行しました。
仮定 4. 一連のログからそれらを出力した関数セットを抽出できる
仮定 5. 処理ルートは関数ペアの呼び出し関係の集まりとして記述できる
仮定 6. 一連のログから調査すべき処理ルートの候補を抽出できる

実施した作業ステップは以下の 7 つです。
ステップ 1: Ceph クラスタのログを回収・管理する仕組みの作成
ステップ 2: ログ行にログ出力箇所を付加する仕組みの作成
ステップ 3: 障害パターン毎に特徴的なログの抽出
ステップ 4: ログレベル設定
ステップ 5: 処理ルートデータベースの作成
ステップ 6: ログ出力箇所を通る処理ルートを検索する仕組みの作成

各ステップについて概要を説明します。

ステップ 1: Ceph クラスタのログを回収・管理する仕組みの作成

Ceph は複数のノードで構成されたクラスタで動作するので、障害調査には、クラスタ単位で同じ時間帯のログが必要とされる場合があります。 性能の観点から、運用時のログレベルは低く設定することが望ましいため、調査に必要とされる詳細なログはメモリログに取得する必要があると考えられます。そこで、クラスタ単位でログの取得開始、終了、回収を一括で行い、ユニークなログ ID に紐づけたログアーカイブで管理する仕組みを作成します。ログの取得終了時にはメモリログのログファイルへのダンプも行います。

ステップ 2: ログ行にログ出力箇所を付加する仕組みの作成

ソースコード上のログ出力箇所を洗い出し、それぞれについてサブシステム、ログレベル、クラス、メソッド、正規表現化したログメッセージを抽出します。次に、実際のログ行に含まれるログメッセージをキーに、各行にログ出力箇所を (クラス::メソッド) の形式で付加するスクリプトに加工します。このスクリプトは、ステップ 1 で取得したログアーカイブに対して実行します。

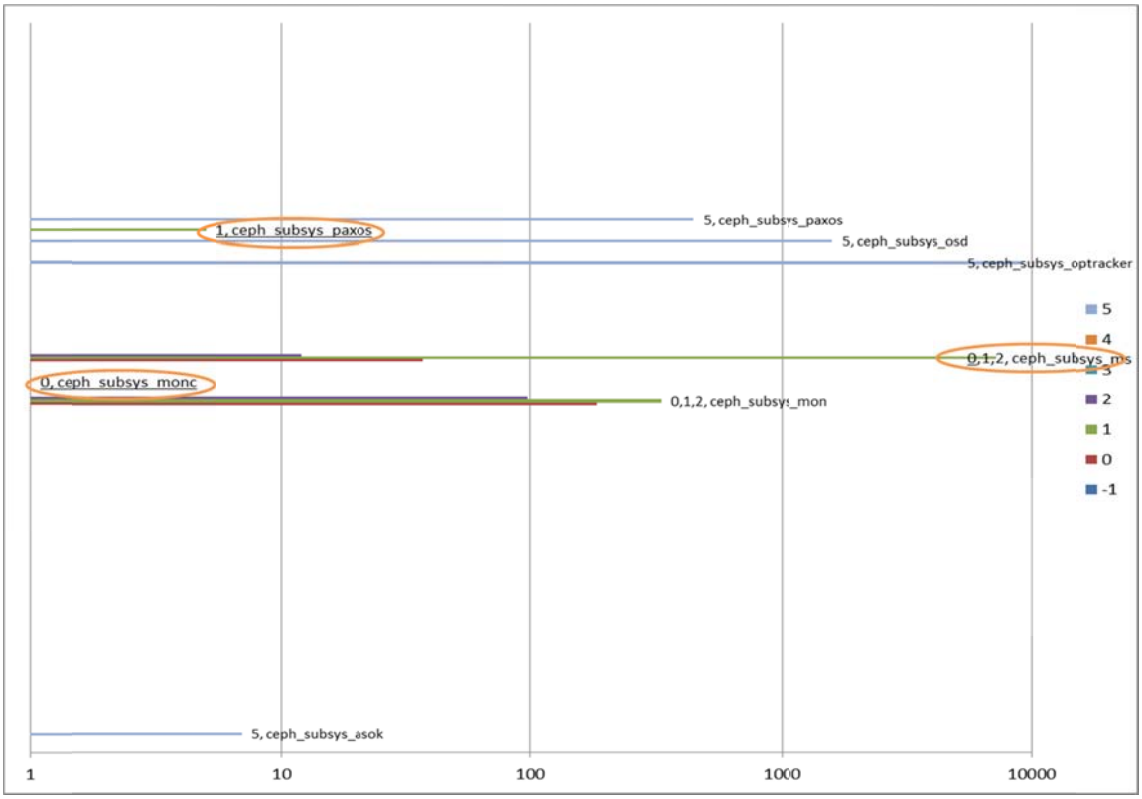
```
7fe4d6eb6700 20 timer(0x7fe4e6cb6070).timer_thread awake        (Timer::timer_thread)
7fe4d6eb6700 10 timer(0x7fe4e6cb6070).timer_thread executing 0x7fe4ebd6dfb0     (Timer::timer_thread)
7fe4d6eb6700  5 osd.1 45634 tick        (OSD::tick)
7fe4d6eb6700 25 osd.1 45634 heartbeat_check osd.0 first_tx 2016-08-23 04:04:28.573642 last_tx 2016-08-24 06:56:01.650271 last_rx_back 2016-08-24 06:56:08.650271
last_rx_front 2016-08-24 06:56:08.650271       (OSD::heartbeat_check)
7fe4d6eb6700 25 osd.2 45634 heartbeat_check osd.2 first_tx 2016-08-24 05:54:57.132631 last_tx 2016-08-24 06:56:01.650271 last_rx_back 2016-08-24 06:56:08.650271
last_rx_front 2016-08-24 06:56:08.650271       (OSD::heartbeat_check)
7fe4d6eb6700 10 osd.1 45634 do_waiters -- start (OSD::do_waiters)
7fe4d6eb6700 10 osd.1 45634 do_waiters -- finish        (OSD::do_waiters)
7fe4d6eb6700 10 timer(0x7fe4e6cb6070).add_event_at 2016-08-24 06:56:15.887674 -> 0x7fe4ebd6d8b0 (Timer::add_event_at)
7fe4d6eb6700 20 timer(0x7fe4e6cb6070).timer_thread going to sleep       (Timer::timer_thread)
7fe4bf30e700 20 OSD::recovery_tp worker waiting (WorkQueue::worker)
7fe4bf30e700 20 heartbeat_map reset_timeout 'OSD::recovery_tp thread 0x7fe4bf30e700' grace 60 suicide 0 (HeartbeatMap::reset_timeout)
7fe4c631c700 20 timer(0x7fff20483048).timer_thread awake        (Timer::timer_thread)
7fe4c631c700 10 timer(0x7fff20483048).timer_thread executing 0x7fe4e6afd940     (Timer::timer_thread)
7fe4c631c700 10 monclient: tick (MonClient::tick,Objecter::tick)
7fe4c631c700 10 cephx: validate_tickets want 37 have 37 need 0  (CephxProtocol::validate_tickets)
7fe4c631c700 20 cephx client: need_tickets: want=37 need=0 have=37      (CephxClientHandler::need_tickets)
7fe4c631c700 10 monclient: _check_auth_rotating have uptodate secrets (they expire after 2016-08-24 06:55:44.088749)    (MonClient::_check_auth_rotating)
7fe4c631c700 10 auth: dump_rotating:    (RotatingKeyRing::dump_rotating)
7fe4c631c700 10 auth:  id 9392 AQCXJr1X3kWTHxAA3HgPX+PboCQ+SqBuKF3x+A== expires 2016-08-24 06:46:15.529726       (RotatingKeyRing::dump_rotating)
7fe4c631c700 10 auth:  id 9393 AQCpNL1XWpq1OBAAfhC1lzorh1kozoEXKkZM0w== expires 2016-08-24 07:46:17.951407       (RotatingKeyRing::dump_rotating)
7fe4c631c700 10 auth:  id 9394 AQC4Qr1Xu+CcDRAA93UBjUm3shKT6x5y8k+eVg== expires 2016-08-24 08:46:17.951407       (RotatingKeyRing::dump_rotating)
7fe4c631c700 10 monclient: renew subs? (now: 2016-08-24 06:56:14.088802; renew after: 2016-08-24 06:58:20.055689) -- no (Locker::handle_client_lease)
7fe4c631c700  1 RefCountedObject::get 0x7fe4ebcf7000 2 -> 3     (RefCountedObj::RefCountedObject)
7fe4c631c700 20 -- 10.100.30.11:6800/14971 send_keepalive con 0x7fe4ebdfa3c0, have pipe.       (SimpleMessenger::send_keepalive)
7fe4c631c700  1 RefCountedObject::put 0x7fe4ebcf7000 3 -> 2     (RefCountedObj::RefCountedObject)
7fe4c631c700 10 timer(0x7fff20483048).add_event_at 2016-08-24 06:56:24.088830 -> 0x7fe4e6afc400 (Timer::add_event_at)
7fe4c631c700 20 timer(0x7fff20483048).timer_thread going to sleep       (Timer::timer_thread)
7fe4baa04700 10 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebdfa3c0).writer: state = open
policy.server=0       (Pipe::writer)
7fe4baa04700 10 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebdfa3c0).write_keepalive2 14
2016-08-24 06:56:14.088858     (Pipe::write_keepalive2)
7fe4baa04700 10 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebdfa3c0).writer: state = open
policy.server=0       (Pipe::writer)
7fe4baa04700 20 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebdfa3c0).writer sleeping
(Pipe::writer)
7fe4b94f6700 20 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebdfa3c0).reader got
KEEPALIVE_ACK   (Pipe::reader)
7fe4b94f6700 20 -- 10.100.30.11:6800/14971 >> 10.100.30.32:6789/0 pipe(0x7fe4ebcf7000 sd=22 :59970 s=2 pgs=157174 cs=1 l=1 c=0x7fe4ebdfa3c0).reader reading
tag...      (Pipe::reader)
7fe4c4b19700 20 OSD::osd_tp worker waiting      (WorkQueue::worker)
7fe4c4b19700 20 heartbeat_map reset_timeout 'OSD::osd_tp thread 0x7fe4c4b19700' grace 60 suicide 0      (HeartbeatMap::reset_timeout)
7fe4c531a700 20 OSD::osd_tp worker waiting      (WorkQueue::worker)
7fe4c531a700 20 heartbeat_map reset_timeout 'OSD::osd_tp thread 0x7fe4c531a700' grace 60 suicide 0      (HeartbeatMap::reset_timeout)
7fe4c0b11700 20 heartbeat_map reset_timeout 'OSD::osd_op_tp thread 0x7fe4c0b11700' grace 15 suicide 150 (HeartbeatMap::reset_timeout)
7fe4c0b11700 20 heartbeat_map reset_timeout 'OSD::osd_op_tp thread 0x7fe4c0b11700' grace 4 suicide 0    (HeartbeatMap::reset_timeout)
7fe4d24a2700 20 FileStore::op_tp worker waiting (WorkQueue::worker)
7fe4d24a2700 20 heartbeat_map reset_timeout 'FileStore::op_tp thread 0x7fe4d24a2700' grace 60 suicide 0 (HeartbeatMap::reset_timeout)
7fe4d2ca3700 20 FileStore::op_tp worker waiting (WorkQueue::worker)
7fe4d2ca3700 20 heartbeat_map reset_timeout 'FileStore::op_tp thread 0x7fe4d2ca3700' grace 60 suicide 0 (HeartbeatMap::reset_timeout)
7fe4da6b6700 20 timer(0x7fe4e6bbb980).timer_thread awake        (Timer::timer_thread)
7fe4da6b6700 10 timer(0x7fe4e6bbb980).timer_thread executing 0x7fe4e6afbf20     (Timer::timer_thread)
7fe4da6b6700 10 osd.1.objecter tick     (MonClient::tick,Objecter::tick)
7fe4da6b6700 10 timer(0x7fe4e6bbb980).add_event_at 2016-08-24 06:56:19.311697 -> 0x7fe4e6afc3d0 (Timer::add_event_at)
7fe4da6b6700 20 timer(0x7fe4e6bbb980).timer_thread going to sleep       (Timer::timer_thread)
```

```
Subsystem(#logs), Loglevel:#logs, ...
ceph_subsys_ms(3565),20:1378,30:992,10:873,1:218,15:102,2:2
ceph_subsys_refs(2729),1:2729
ceph_subsys_optracker(1064),5:1064
ceph_subsys_mon(1016),20:552,10:322,30:32,15:31,7:30,2:28,0:9,11:5,1:5,25:2
ceph_subsys_paxos(576),10:444,5:56,20:49,7:27
ceph_subsys_auth(281),10:186,20:83,30:12
ceph_subsys_throttle(186),10:186
ceph_subsys_timer(162),10:110,20:52
ceph_subsys_rbd(61),20:61
ceph_subsys_mds(59),10:59
ceph_subsys_finisher(44),10:44
ceph_subsys_monc(40),10:40
ceph_subsys_asok(5),30:4,5:1
ceph_subsys_osd(4),10:4
ceph_subsys_filestore(3),10:3
```

上記の 2 つの図はスクリプトを実行した結果の一部です。各ログ行にログ出力箇所が付加される他、サブシステムおよびログレベル毎のログ出力回数をカウントします。

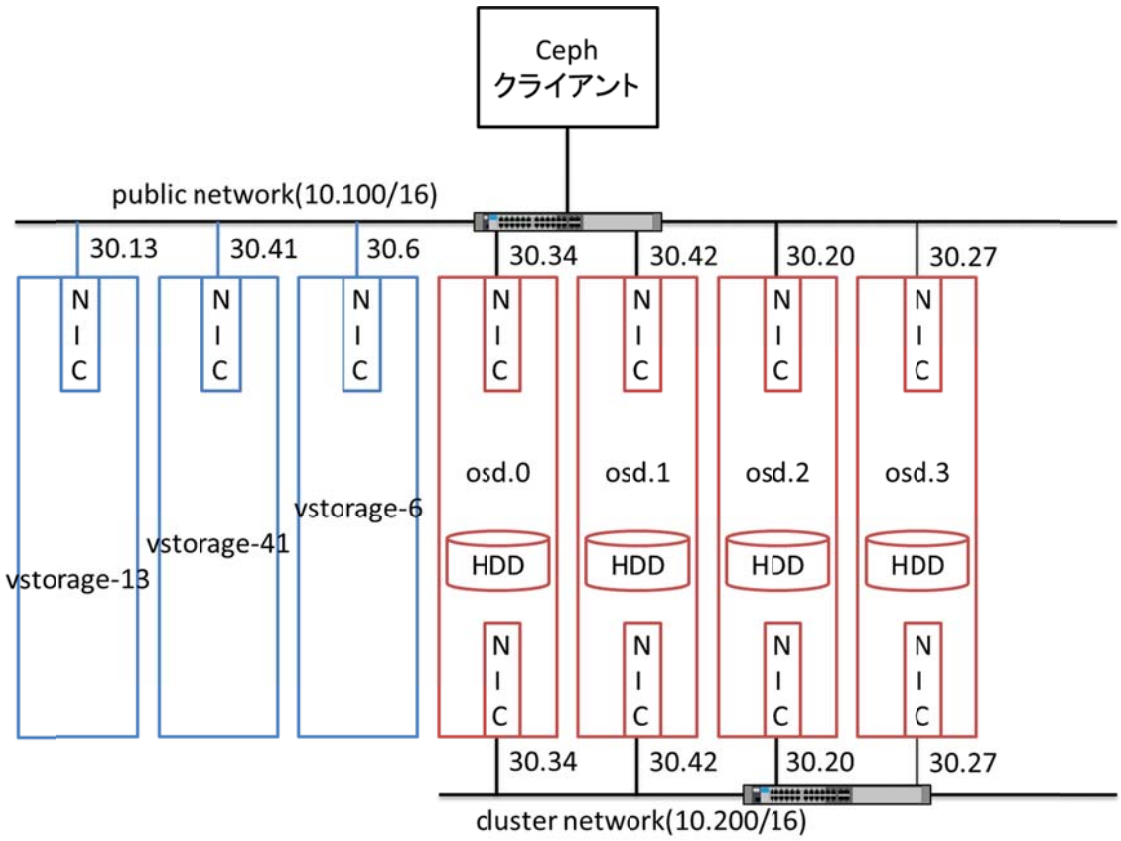
ステップ 3: 障害パターン毎の特徴なログの抽出

OpenStack/Ceph 異常系テストの実施の際にステップ 1 の仕組みで障害パターン毎のログアーカイブを取得します。各ログアーカイブに対してステップ 2 のスクリプトを実行することで、障害パターン毎のログのサブシステムおよびログレベルの分布を抽出します。各障害パターンのログレベル分布を、障害がない状態で同じオペレーションを実行したときにログレベル分布と比較することで、その障害パターンによって出力された可能性が高いログを抽出します。



上記の図は、OpenStack/Ceph 異常系テストのテスト項目 (故障パターン) のうち、テスト項目 #1 の通常の状態と、テスト項目 #34 の 3 台中 2 台の Ceph MON でダウンが発生した場合でのログレベル分布を比較した例です。丸で囲ったサブシステムとログレベルのログがテスト項目 #34 でのみ出力されていたことを表します。

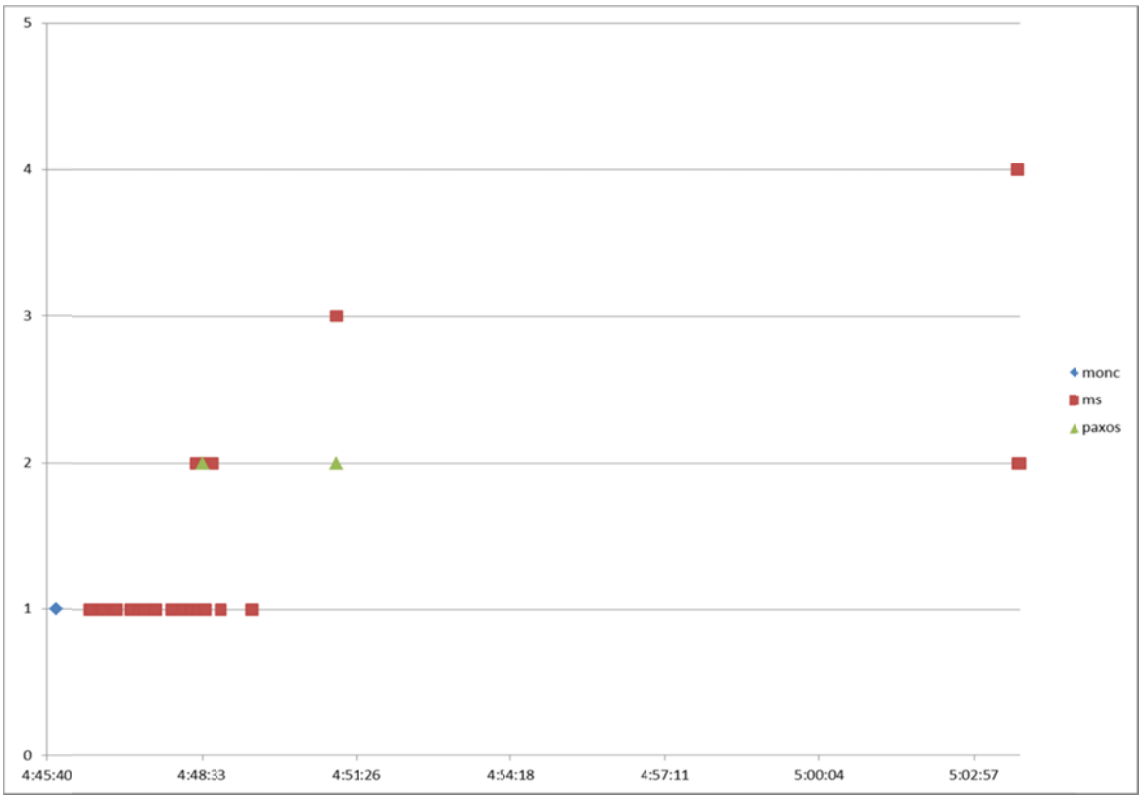
「3.4. テスト結果」で述べたとおり、テスト項目 #34 では、コマンド (ceph –watch) の出力から Ceph MON がダウンしていることが判別できませんでした。



上記の図はテスト項目を実施したときの Ceph クラスタの構成です。

具体的には、2 台の Ceph MON がダウンしたにも関わらず、コマンド (ceph –watch) の出力上では、1 台の Ceph MON が停止し HELTH_WARN に状態遷移し、当該 Ceph MON 再起動後に HEALTH_OK に状態したように見え、もう 1 台の Ceph MON が停止したことが分かりませんでした。

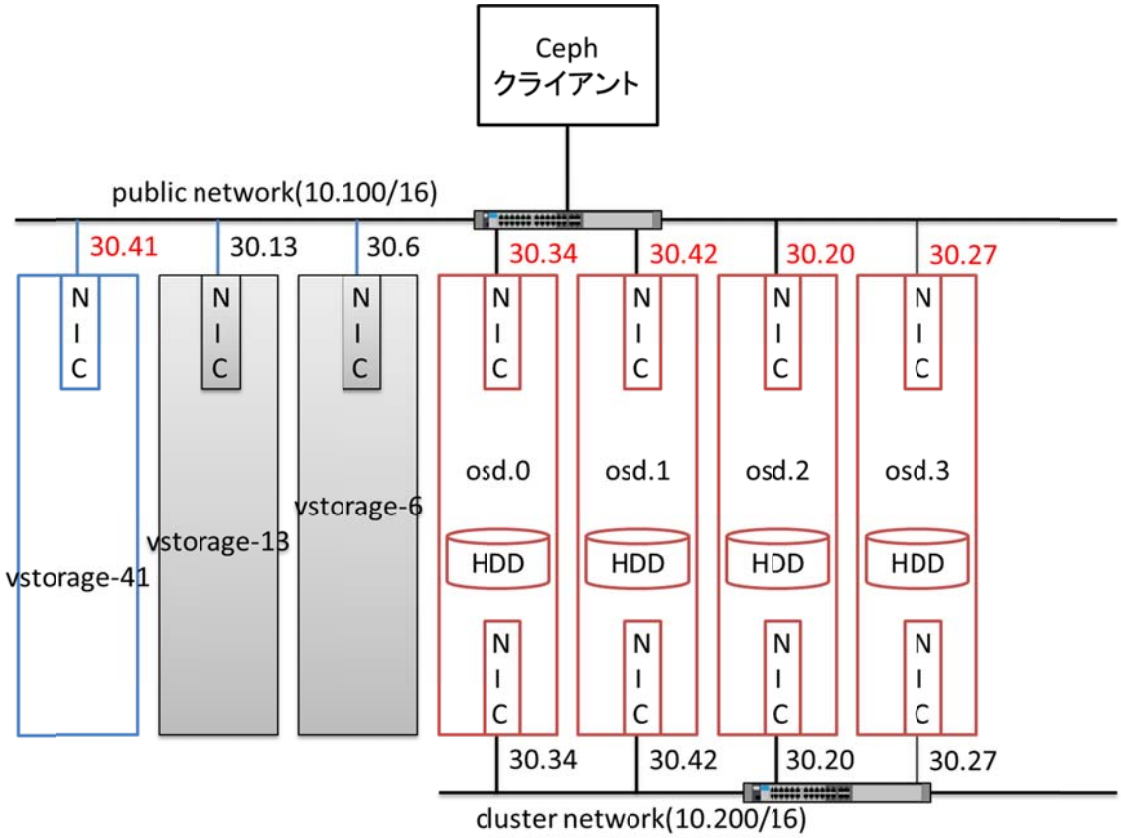
Ceph MON クラスタは奇数台数で構成され、その動作には多数派を形成できる台数 (＝定足数) が健全である必要があります。テスト項目 #34 では、3 台で構成された Ceph MON クラスタのうち 2 台がダウンしてしまうため、健全な Ceph MON の台数＝1 が定足数＝2 に満たない状況となります。また、コマンド (ceph –watch) の動作は Ceph MON クラスタが動作していることに依存します。よって、Ceph MON クラスタの定足数がない状態で発生した、もう 1 台の Ceph MON が停止、再起動の事象を、コマンド (ceph –watch) が正常に動作せず検知できなかったものと思われます。



上記の図は、テスト項目 #1 の通常の状態ログレベル分布を比較して抽出された、テスト項目 #34 でのみ出力されていたログの時系列での件数分布です。

ceph-mon.vstorage-41.log	4:45:40	1	(Paxos::lease_timeout)
ceph-osd.2.log	4:45:50	0	(MonClient::ms_handle_reset)
ceph-osd.3.log	4:46:30	0	(Pipe::fault)
ceph-osd.1.log	4:46:32	0	(Pipe::fault)
ceph-osd.2.log	4:46:40	0	(Pipe::fault)
ceph-osd.1.log	4:46:42	0	(Pipe::fault)
ceph-osd.0.log	4:46:44	0	(Pipe::fault)
ceph-osd.2.log	4:46:55	0	(Pipe::fault)
ceph-osd.0.log	4:46:59	0	(Pipe::fault)
ceph-osd.3.log	4:47:15	0	(Pipe::fault)
ceph-osd.2.log	4:47:25	0	(Pipe::fault)
ceph-osd.3.log	4:47:30	0	(Pipe::fault)
ceph-osd.1.log	4:47:41	0	(Pipe::fault)
ceph-osd.3.log	4:48:00	0	(Pipe::fault)
ceph-osd.1.log	4:48:11	0	(Pipe::fault)
ceph-osd.2.log	4:48:22	0	(Pipe::fault)
ceph-mon.vstorage-41.log	4:48:26	0	(AsyncConection::fault)
ceph-mon.vstorage-41.log	4:48:26	0	(Pipe::fault)
ceph-osd.0.log	4:48:29	0	(Pipe::fault)
ceph-mon.vstorage-13.log	4:48:33	1	(PaxosService::refresh)
ceph-mon.vstorage-13.log	4:48:33	1	(PaxosService::refresh)
ceph-mon.vstorage-41.log	4:48:34	0	(Pipe::accept)
ceph-mon.vstorage-41.log	4:48:34	0	(Pipe::accept)
ceph-mon.vstorage-13.log	4:48:37	0	(Pipe::fault)
ceph-osd.0.log	4:48:44	0	(Pipe::accept)
ceph-osd.0.log	4:48:44	0	(Pipe::accept)
ceph-osd.2.log	4:48:46	0	(Pipe::accept)
ceph-osd.2.log	4:48:46	0	(Pipe::accept)
ceph-osd.2.log	4:48:55	0	(Pipe::fault)
ceph-osd.0.log	4:49:29	0	(Pipe::fault)
ceph-mon.vstorage-6.log	4:51:03	1	(PaxosService::refresh)
ceph-mon.vstorage-6.log	4:51:04	1	(PaxosService::refresh)
ceph-mon.vstorage-13.log	4:51:04	0	(Pipe::accept)
ceph-mon.vstorage-41.log	4:51:04	0	(Pipe::accept)
ceph-mon.vstorage-41.log	4:51:04	0	(Pipe::accept)

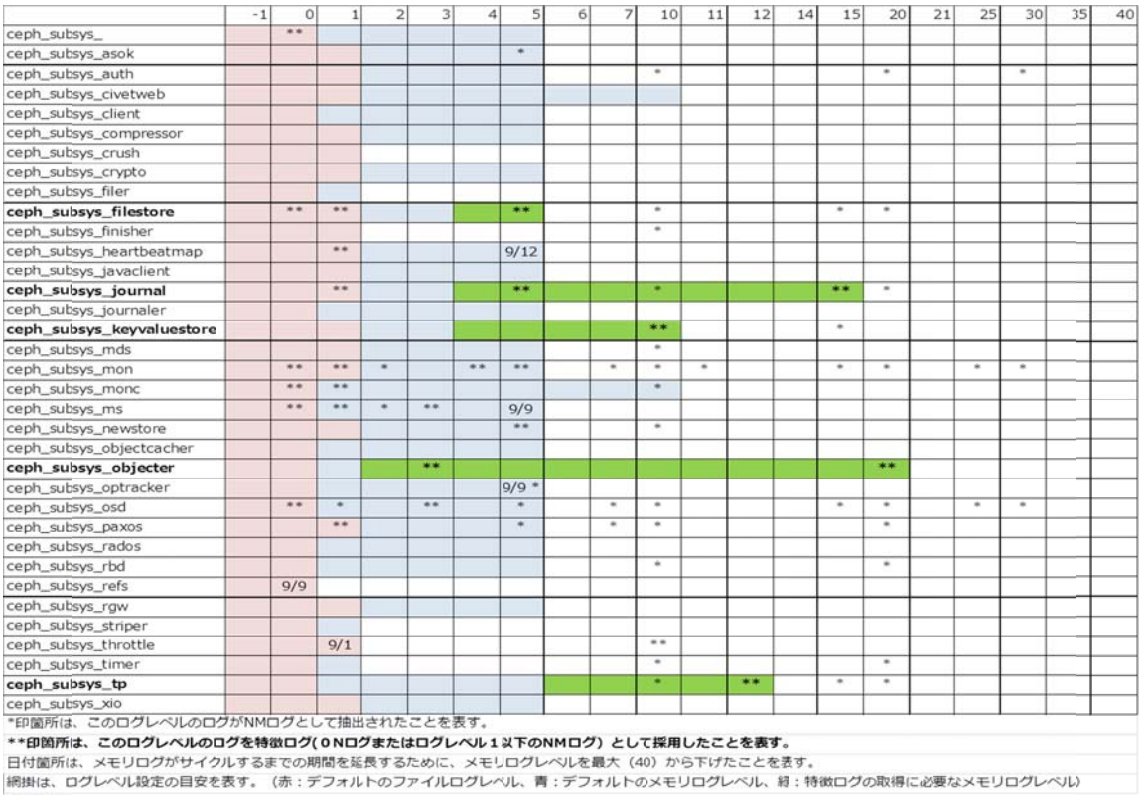
上記の図は、テスト項目 #1 の通常の状態とのログレベル分布を比較して抽出された、
テスト項目 #34 でのみ出力されていたログ行です。赤線で囲った部分は、2 台の Ceph MON への通信が、その他全てのノード上で失敗している状況を表示しています。



上記の図は、テスト項目 #34 でダウンさせた Ceph MON 2台を灰色で示し、テスト項目 #1 の通常の状態のログレベル分布を比較してテスト項目 #34 でのみ出力されていたログにある通信失敗の箇所を赤文字で示しています。つまり、テスト項目 #34 の実施によって検出された、コマンド (ceph –watch) で現象 (3 台中 2 台の Ceph MON ダウン) を検知できない問題に対する代替手段として、故障パターンに特徴的なログの内容 (通信が失敗している箇所) から現象を推測する方法が確認できたと思います。

4.2. Ceph ログの問題点への対処 (つづき)

ステップ 4: ログレベル設定
全てのサブシステムおよびログレベルの組み合わせのうち、ステップ 3 で抽出した障害パターンに特徴的なログを出力したサブシステムおよびログレベルの組み合わせを抽出します。
抽出されたサブシステムおよびログレベルの総和でログレベルを設定します。
このログレベル設定でログ頻度が高くなり過ぎてメモリログが短時間でサイクルしてしまう場合は、
ログ出力頻度の高い箇所を避けるようにログレベルを調整することでメモリ上にログが残る時間を延長します。



	-1	0	1	2	3	4	5	6	7	10	11	12	14	15	20	21	25	30	35	40
ceph_subsys_		**																		
ceph_subsys_asok							*													
ceph_subsys_auth										*					*			*		
ceph_subsys_civetweb																				
ceph_subsys_client																				
ceph_subsys_compressor																				
ceph_subsys_crush																				
ceph_subsys_crypto																				
ceph_subsys_filer																				
ceph_subsys_filestore		**	**				**			*				*	*					
ceph_subsys_finisher										*										
ceph_subsys_heartbeatmap			**				9/12													
ceph_subsys_javaclient																				
ceph_subsys_journal			**				**			*				**	*					
ceph_subsys_journaler																				
ceph_subsys_keyvaluestore										**				*						
ceph_subsys_mds										*										
ceph_subsys_mon		**	**	*		**	**		*	*	*			*	*		*	*		
ceph_subsys_monc		**	**							*										
ceph_subsys_ms		**	**	*	**		9/9													
ceph_subsys_newstore							**			*										
ceph_subsys_objectcacher																				
ceph_subsys_objecter					**										**					
ceph_subsys_optracker							9/9 *													
ceph_subsys_osd		**	*		**		*		*	*				*	*		*	*		
ceph_subsys_paxos			**				*		*	*					*					
ceph_subsys_rados																				
ceph_subsys_rbd										*					*					
ceph_subsys_refs		9/9																		
ceph_subsys_rgw																				
ceph_subsys_striper																				
ceph_subsys_throttle			9/1							**										
ceph_subsys_timer										*					*					
ceph_subsys_tp										*		**		*	*					
ceph_subsys_xio																				

*印箇所は、このログレベルのログがNMログとして抽出されたことを表す。
**印箇所は、このログレベルのログを特徴ログ（０Nログまたはログレベル１以下のNMログ）として採用したことを表す。
日付箇所は、メモリログがサイクルするまでの期間を延長するために、メモリログレベルを最大（40）から下げたことを表す。
網掛は、ログレベル設定の目安を表す。（赤：デフォルトのファイルログレベル、青：デフォルトのメモリログレベル、緑：特徴ログの取得に必要なメモリログレベル）

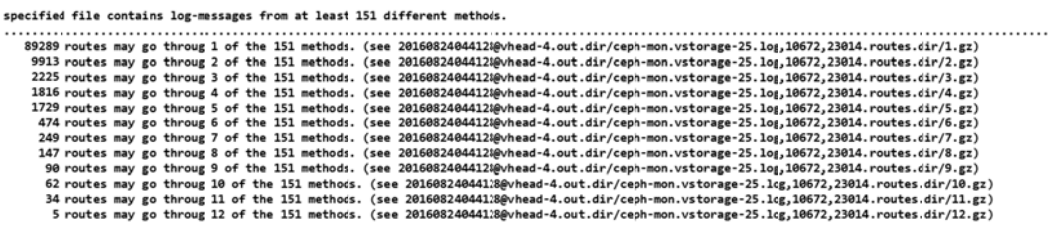
図は、「5. OpenStack/Ceph 異常系テスト (2) 」で述べるテストを通して得られたログから抽出したログレベル設定です。緑色のマーク部分はデフォルトのメモリログレベルから変更していることを表します。

ステップ 5: 処理ルートデータベースの作成

ソースコード上の関数呼び出し箇所を洗い出し、それぞれについて呼び出し元クラス::メソッド、呼び出し先クラス::メソッドを抽出します。次に、処理ルートを抽出し、処理ルートデータベースとします。
処理ルートの抽出は、呼び出されることのないクラス::メソッドから何も呼び出さないクラス::メソッドに至るまでの一連の関数呼び出し関係を機械的に検索することで行います。
関数呼び出しを行うかどうかは多くの場合に条件がありますが、この機械的な検索において条件は考慮しません。つまり、関数呼び出しの条件は常に真です。
よって処理ルートデータベースには実際には通ることのない処理ルートが多数含まれます。

ステップ 6: ログ出力箇所を通る処理ルートを検索する仕組みの作成

任意のログに対してステップ 2 のスクリプトを実行することにより、ログに含まれているログ出力箇所のリストが得られます。
これらのログ出力箇所のうちの 1 個から N 個を含むような処理ルートをステップ 5 の処理ルートデータベースから検索します。



```
specified file contains log-messages from at least 151 different methods.
.....................................................................................................................................................
   89289 routes may go throug 1 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/1.gz)
    9913 routes may go throug 2 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/2.gz)
    2225 routes may go throug 3 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/3.gz)
    1816 routes may go throug 4 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/4.gz)
    1729 routes may go throug 5 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/5.gz)
     474 routes may go throug 6 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/6.gz)
     249 routes may go throug 7 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/7.gz)
     147 routes may go throug 8 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/8.gz)
      90 routes may go throug 9 of the 151 methods. (see 20160824044121@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/9.gz)
      62 routes may go throug 10 of the 151 methods. (see 20160824044[illegible]8@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/10.gz)
      34 routes may go throug 11 of the 151 methods. (see 20160824044[illegible]8@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/11.gz)
       5 routes may go throug 12 of the 151 methods. (see 20160824044[illegible]8@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/12.gz)
```

上記の図は、処理ルートデータベースを検索した結果のうち、検索で見つかった処理ルートの数の表示です。

```
$ zcat 20160824044128@vhead-4.out.dir/ceph-mon.vstorage-25.log,10672,23014.routes.dir/12.gz | loglevel.sh
64200 Monitor::ms_dispatch - ((Monitor::_ms_dispatch)) - Monitor::dispatch_op - Elector::dispatch - Elector::handle_propose - (Elector::defer) -
Elector::reset_timer - Elector::C_ElectionExpire::finish - (Elector::expire) - Elector::victory - Monitor::win_election - (Monitor::finish_election) -
(Monitor::resend_routed_requests) - PaxosService::C_RetryMessage::_finish - (PaxosService::dispatch) - MDSMonitor::prepare_update - MDSMonitor::prepare_beacon -
PaxosService::request_proposal - ((PaxosService::propose_pending)) - ((Paxos::trigger_propose)) - ((Paxos::propose_pending)) - ((Paxos::begin)) -
((Paxos::commit_start)) - PaxosService::C_Committed::finish - ((Paxos::commit_finish)) - (Paxos::do_refresh) - ((Monitor::refresh_from_paxos)) -
((PaxosService::refresh)) - ((LogMonitor::update_from_paxos)) - ((LogMonitor::check_subs)) - (LogMonitor::check_sub) - (LogMonitor::_create_sub_summary)
64201 Monitor::ms_dispatch - ((Monitor::_ms_dispatch)) - Monitor::dispatch_op - Elector::dispatch - Elector::handle_propose - (Elector::defer) -
Elector::reset_timer - Elector::C_ElectionExpire::finish - (Elector::expire) - Elector::victory - Monitor::win_election - (Monitor::finish_election) -
(Monitor::resend_routed_requests) - PaxosService::C_RetryMessage::_finish - (PaxosService::dispatch) - MDSMonitor::prepare_update - MDSMonitor::prepare_beacon -
PaxosService::request_proposal - ((PaxosService::propose_pending)) - ((Paxos::trigger_propose)) - ((Paxos::propose_pending)) - ((Paxos::begin)) -
((Paxos::commit_start)) - PaxosService::C_Committed::finish - ((Paxos::commit_finish)) - (Paxos::do_refresh) - ((Monitor::refresh_from_paxos)) -
((PaxosService::refresh)) - ((LogMonitor::update_from_paxos)) - ((LogMonitor::check_subs)) - (LogMonitor::check_sub) - (RefCountedObject::put)
64211 Monitor::ms_dispatch - ((Monitor::_ms_dispatch)) - Monitor::dispatch_op - Elector::dispatch - Elector::handle_propose - (Elector::defer) -
Elector::reset_timer - Elector::C_ElectionExpire::finish - (Elector::expire) - Elector::victory - Monitor::win_election - (Monitor::finish_election) -
(Monitor::resend_routed_requests) - PaxosService::C_RetryMessage::_finish - (PaxosService::dispatch) - MDSMonitor::prepare_update - MDSMonitor::prepare_beacon -
PaxosService::request_proposal - ((PaxosService::propose_pending)) - ((Paxos::trigger_propose)) - ((Paxos::propose_pending)) - ((Paxos::begin)) -
((Paxos::commit_start)) - PaxosService::C_Committed::finish - ((Paxos::commit_finish)) - (Paxos::do_refresh) - ((Monitor::refresh_from_paxos)) -
((PaxosService::refresh)) - ((PGMonitor::update_from_paxos)) - ((PGMonitor::apply_pgmap_delta)) - (MonitorDBStore::get)
64214 Monitor::ms_dispatch - ((Monitor::_ms_dispatch)) - Monitor::dispatch_op - Elector::dispatch - Elector::handle_propose - (Elector::defer) -
Elector::reset_timer - Elector::C_ElectionExpire::finish - (Elector::expire) - Elector::victory - Monitor::win_election - (Monitor::finish_election) -
(Monitor::resend_routed_requests) - PaxosService::C_RetryMessage::_finish - (PaxosService::dispatch) - MDSMonitor::prepare_update - MDSMonitor::prepare_beacon -
PaxosService::request_proposal - ((PaxosService::propose_pending)) - ((Paxos::trigger_propose)) - ((Paxos::propose_pending)) - ((Paxos::begin)) -
((Paxos::commit_start)) - PaxosService::C_Committed::finish - ((Paxos::commit_finish)) - (Paxos::do_refresh) - ((Monitor::refresh_from_paxos)) -
((PaxosService::refresh)) - ((PGMonitor::update_from_paxos)) - ((PGMonitor::read_pgmap_full)) - ((PGMonitor::read_pgmap_meta)) - (MonitorDBStore::get)
64215 Monitor::ms_dispatch - ((Monitor::_ms_dispatch)) - Monitor::dispatch_op - Elector::dispatch - Elector::handle_propose - (Elector::defer) -
Elector::reset_timer - Elector::C_ElectionExpire::finish - (Elector::expire) - Elector::victory - Monitor::win_election - (Monitor::finish_election) -
(Monitor::resend_routed_requests) - PaxosService::C_RetryMessage::_finish - (PaxosService::dispatch) - MDSMonitor::prepare_update - MDSMonitor::prepare_beacon -
PaxosService::request_proposal - ((PaxosService::propose_pending)) - ((Paxos::trigger_propose)) - ((Paxos::propose_pending)) - ((Paxos::begin)) -
((Paxos::commit_start)) - PaxosService::C_Committed::finish - ((Paxos::commit_finish)) - (Paxos::do_refresh) - ((Monitor::refresh_from_paxos)) -
((PaxosService::refresh)) - ((PGMonitor::update_from_paxos)) - ((PGMonitor::read_pgmap_meta)) - (MonitorDBStore::get)
debug mon=20/20
debug paxos=10/10
debug refs=1/1
```

上記の図は、処理ルートデータベースを検索した結果のうち、12 のログ出力箇所を通る 5 つの処理ルートに関する内容です。

ログから障害の切り分けを行うことを想定した場合の、現状の Ceph ログの問題点とその対処案を検討については以上です。

この後に述べる OpenStack/Ceph 異常系テスト (2) においては、対処方法の情報として、ログだけからハード障害を切り分けできるかどうかという観点を加えています。

5. OpenStack/Ceph 異常系テスト (2)

OpenStack 環境のストレージを Ceph に集約することによる課題への取り組みとして、与えられたシステム構成で想定されるハードウェア障害が発生した際の対処に必要な情報を収集・整理します。

障害が及ぼす様々な影響のうち、「3. OpenStack/Ceph 異常系テスト (1)」ではアプリケーションの I/O への影響に着目しました。ここから説明する OpenStack/Ceph 異常系テスト (2) ではシステム管理者の操作への影響に着目します。
システム管理者が OpenStack 環境に対してオペレーションを行っている状況で Ceph を構成するいずれかのハードウェアで故障を発生させます。この状況から、故障したハードウェアを交換するために必要な一連の手順を行います。

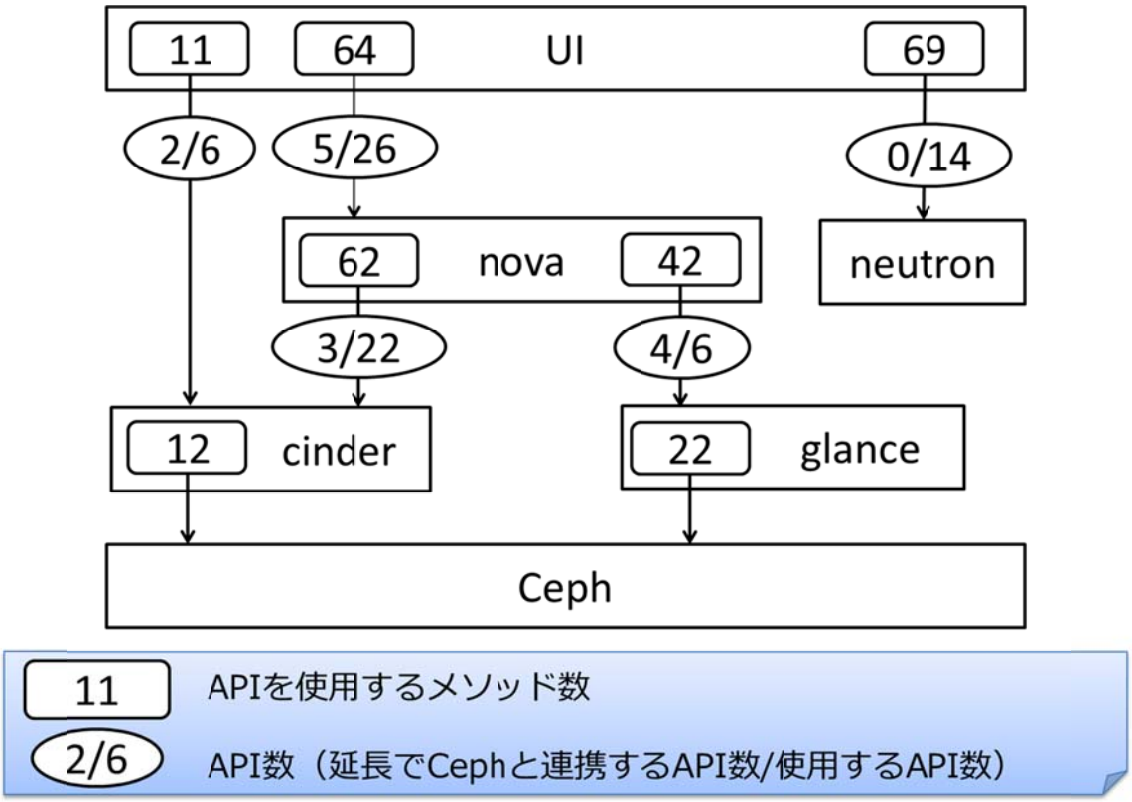
OpenStack/Ceph 異常系テスト (1) では、テスト項目はハードウェア障害パターンに対応するもので、テスト項目数は 53 項目でした。OpenStack/Ceph 異常系テスト (2) では、テスト項目はハードウェア故障パターンとシステム管理者が OpenStack 環境に対して実施するオペレーションの組み合わせに対応します。
オペレーションは 9 種類を設定しました。各オペレーションについては後述します。
53 個のハードウェア障害パターンのうち、OpenStack 環境に対するオペレーションへの影響がないことが明らかな 2 パターン (#52 と #53) とデータ冗長度 5 の設定の 1 パターン (#21) は組み合わせから除外しました。後者については、システム管理者の操作への影響という観点から、データ冗長度は 4 までの設定 (#20) で十分と考えたことが除外した主な理由です。よって、テスト項目数は 50 x 9＝450 項目です。

OpenStack/Ceph 異常系テスト (1) では、結果的に、コマンド出力等では障害箇所の特定ができない場合がありました。すなわち、コマンド (ceph -watch) 出力でハードウェア障害のイベント発生が分からないことや、コマンド (ceph -watch) 出力と実際のイベントが一致しないことがあるという点です。
フィールドサポートの場面としては障害箇所の特定のためのオペレーション (コマンド実行等) をタイムリーに実施することが困難な場合も想定されます。
このような場合に対しては取得済みの Ceph のログからの障害箇所の特定が必要と思われます。
上記問題のフィードバックとして、OpenStack/Ceph 異常系テスト (2) では、ログだけからハード障害を切り分けできるかどうかという観点を加えています。OpenStack/Ceph 異常系テスト (2) ではテスト項目毎に、故障前、故障中、復旧後の時間的範囲での Ceph のログを取得します。

5.1. システム管理者が OpenStack 環境に対して実施するオペレーションの選択

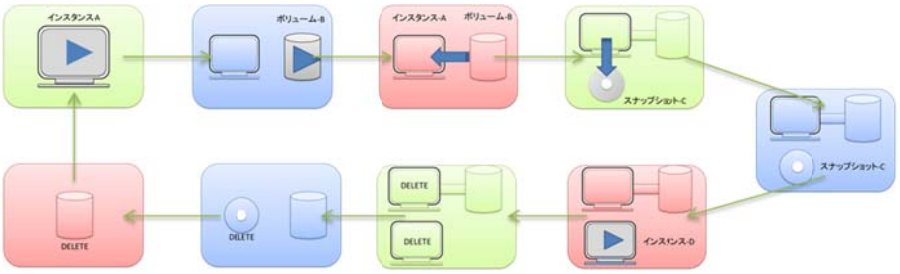
OpenStack/Ceph 異常系テスト (2) では、テスト項目はハードウェア故障パターンとシステム管理者が OpenStack 環境に対して実施するオペレーションの組み合わせに対応します。
オペレーションは 9 種類を設定しました。オペレーションの設定は以下のステップで実施しました。
ステップ 1:システム管理者が使用する OpenStack API をリストアップ
ステップ 2:うち、延長で Ceph 連携のある OpenStack API をリストアップ

ステップ 3:リストアップした OpenStack API を使用して循環するオペレーションのシーケンスを作成



上記の図がステップ 1 およびステップ 2 のリストアップの結果です。

直接使用する OpenStack API が 46 個ありました。うち、延長で Ceph 連携のあるものは 7 個でした。



通番	Ceph連携	対象API	手順
api-1	nova-17	servers.create	既存のイメージからインスタンス-A を起動
api-2	cinder-3	volumes.create	新規ボリューム-B を作成
api-3	nova-25	volumes.create_server_volume	ボリューム-B をインスタンス-A に接続
api-4	nova-8 nova-17	images.get servers.create	インスタンス-A のスナップショット-C を作成
api-5	nova-8	images.get	スナップショット-C の情報を取得して、スナップショット-C がACTIVE になるのを待機
api-6	nova-17	servers.create	スナップショット-C から新たなインスタンス-D を起動
api-7	nova-18	servers.delete	起動したインスタンス-A、インスタンス-D を削除
api-8	nova-7	images.delete	スナップショット-C を削除
api-9	cinder-4	volumes.delete	ボリューム-B を削除

上記の図および表はステップ 3 で作成したシーケンスです。
シーケンスは api-1 から api-9 の 9 個のオペレーションで構成されます。api-9 のオペレーションを実行すると、api-1 のオペレーションを実行する前の、最初の状態に戻ります。

api-1: 既存のイメージからインスタンス-A を起動
api-2: 新規ボリューム-B を作成
api-3: ボリューム-B をインスタンス-A に接続
api-4: インスタンス-A のスナップショット-C を作成
api-5: スナップショット-C の情報を取得して、スナップショット-C が ACTIVE になるのを待機
api-6: スナップショット-C から新たなインスタンス-D を起動
api-7: 起動したインスタンス-A、インスタンス-D を削除
api-8: スナップショット-C を削除
api-9: ボリューム-B を削除

5.2. テスト項目実施方法

OpenStack/Ceph 異常系テスト (2) は OpenStack/Ceph 異常系テスト (1) と基本的に同じシステムを使用して実施しますが、テスト実施時期の違いから使用するソフトウェアバージョンに以下の違いがあります。

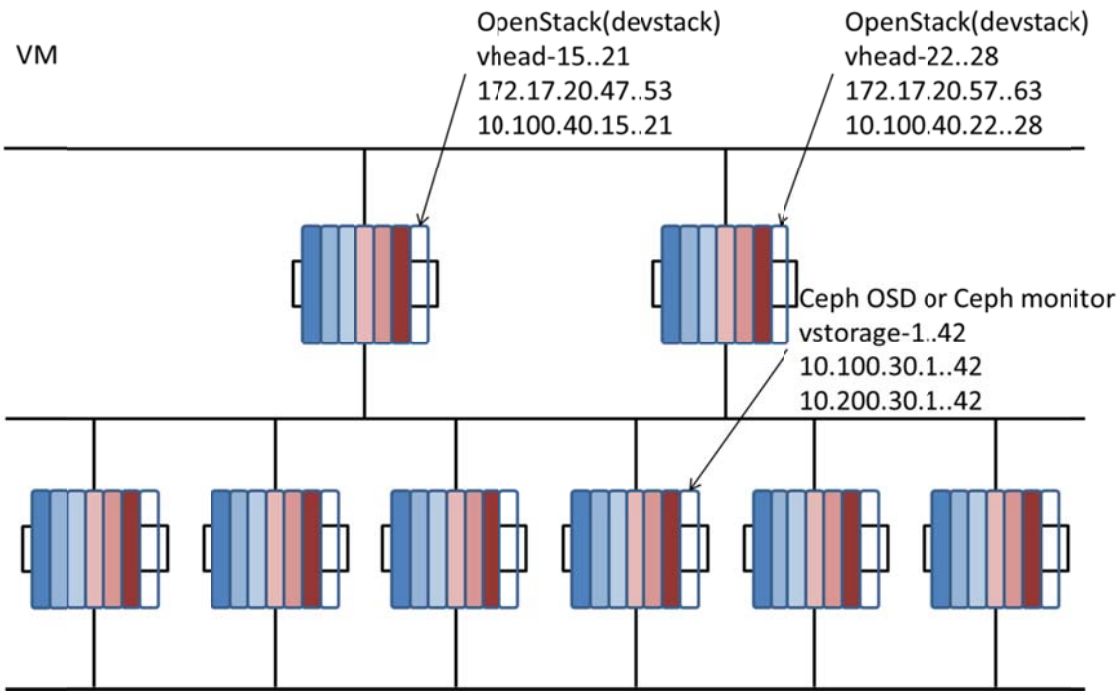
OpenStack/Ceph 異常系テスト (1) で使用したソフトウェアのバージョン:

- OS: Ubuntu 14.04.2 LTS
- Ceph: hammer (v0.94.1)
- OpenStack: stable/kilo

OpenStack/Ceph 異常系テスト (2) 使用するソフトウェアのバージョン:

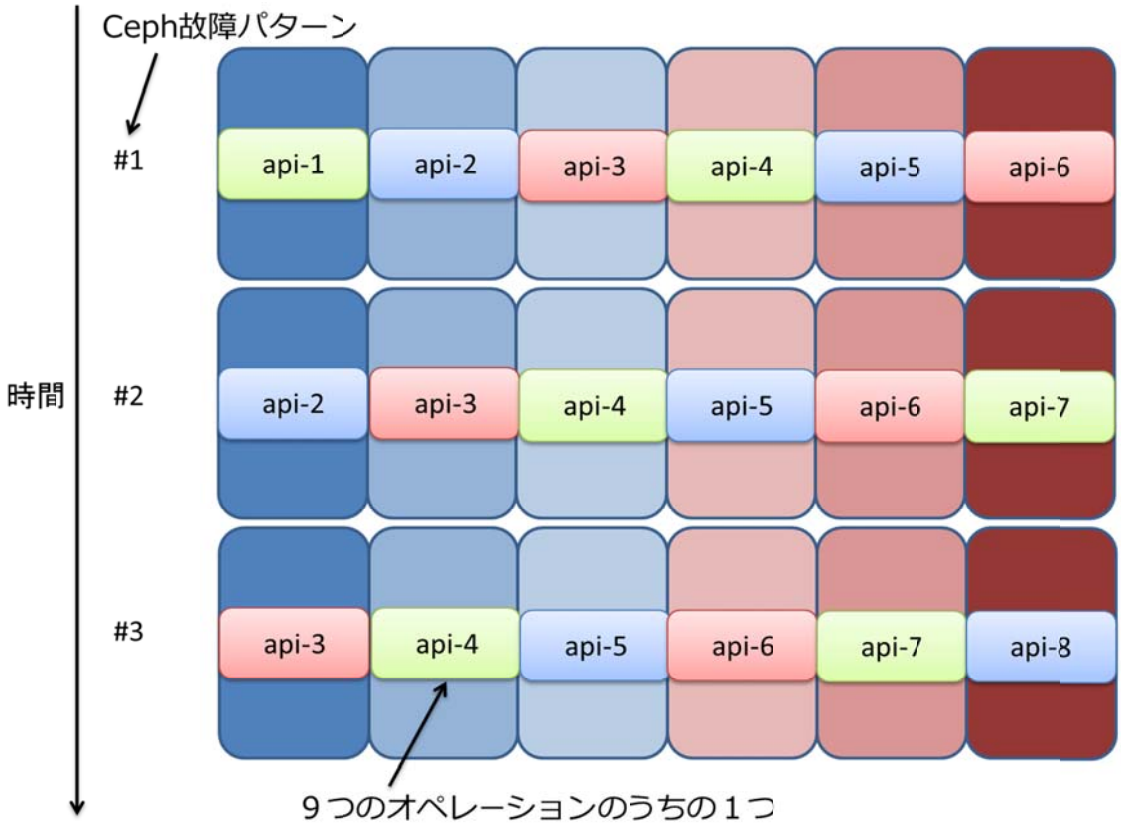
- OS: Ubuntu 14.04.4 LTS
- Ceph: infernalis (v9.2.0)
- OpenStack: stable/liberty

「3.1. システム構成」で述べたように、テストは同じ構成のシステムを複数セット使用して並行実施します。



上記の図は、物理的サーバーの構成と、その上に VM で構成した 7 つのシステムを表しています。

OpenStack/Ceph 異常系テスト (2) は項目数が 450 項目と、OpenStack/Ceph 異常系テスト (1) の 53 項目に比べて格段に多いため、実施にあたってはテスト全体に掛かる時間を短縮するための工夫が必要でした。api-1 から api-9 の 9 個のオペレーションが循環するのもその工夫の一つです。これは、あるテスト項目の実行が次のテスト項目の準備を兼ねるようにすることで、テスト項目と次のテスト項目の間に必要な手順をなるべく減らすことを意図しています。



上記の図は、7 システムのうちの 6 システムを使用して、6 個のテスト項目を並行に実施する様子を表しています。

6 個のシステムで同じ障害パターンを発生させ、6 個のオペレーションへの影響を同時に観測しています。7 個目のシステムについては、必要な Ceph OSD 台数の多い故障パターンで、他の 6 システムに Ceph OSD を融通するためのバッファとしています。オペレーションは 9 個あるので、9 個のシステムが使用できることが作業効率上は望ましいですが、リソース上の制約により 6 並列までとしています。

5.3. ログの分析

ログの取得には「4.2. Ceph ログの問題点への対処」で述べた Ceph クラスタのログを回収・管理する仕組みを使用し、テスト項目実施毎にテスト ID をログアーカイブに紐づけて管理します。また、取得したログアーカイブはログ行にログ出力箇所を付加する仕組みを使用してログ出力箇所のサブシステムおよびログレベルの分布を取得します。
各項目のサブシステムおよびログレベル分布は、故障のない状態 (障害パターン#1) で同じオペレーションを実行したときのサブシステムおよびログレベル分布と比較することで、故障時にのみ出力された、あるいは、出力頻度の増加が顕著なログを抽出します。前者を 0N ログ (ゼロエヌログ) と呼びます。これはログの出力頻度が 0 から N に上がったログという意味です。後者を NM ログ (エヌエムログ) と呼びます。これはログの出力頻度が N から M に上がったという意味です。ここでの N は 0 を含みます。
上述したように、故障パターン毎に 9 個のオペレーションを実施します。故障パターンに固有のサブシステムおよびログレベル分布があると仮定し、9 個のオペレーションに共通して抽出されるログを特徴ログとします。ログレベル設定を高く設定すると、ログレベル設定が低い場合に比べて抽出される 0N ログおよび NM ログの量は多くなります。運用上、より低いログレベル設定で (＝より少ない量のログから) 特徴ログが検出できることが望ましいので、NM ログはログレベルの値を 0 か 1 に限定することにします。0N ログに関しては、故障時にのみ出力されたログであり、障害パターンに由来している可能性が高いのでログレベルの範囲は限定しませんが、ON ログの量が多くなりすぎる場合はログレベル を 0 か 1 に限定し、NM ログを特徴ログに選択することにします。

OpenStack/Ceph 異常系テスト (2) ではテスト項目毎に、故障前、故障中、復旧後の時間的範囲での Ceph のログを取得します。よって、抽出された特徴ログ (ON ログまたはログレベル 1 以下の NM ログ) には障害発生に由来するログだけでなく、障害復旧のために行ったオペレーションに由来するログも含まれる可能性があります。ログだけからハード障害を切り分けできるかどうかという観点から、障害復旧に由来するログはできるだけ除外します。
そこで、抽出された特徴ログ (ON ログまたはログレベル 1 以下の NM ログ) は、ログアーカイブにログが記録されている時間的範囲の冒頭の 1 分間に含まれていたものとそれ以外を区別します。

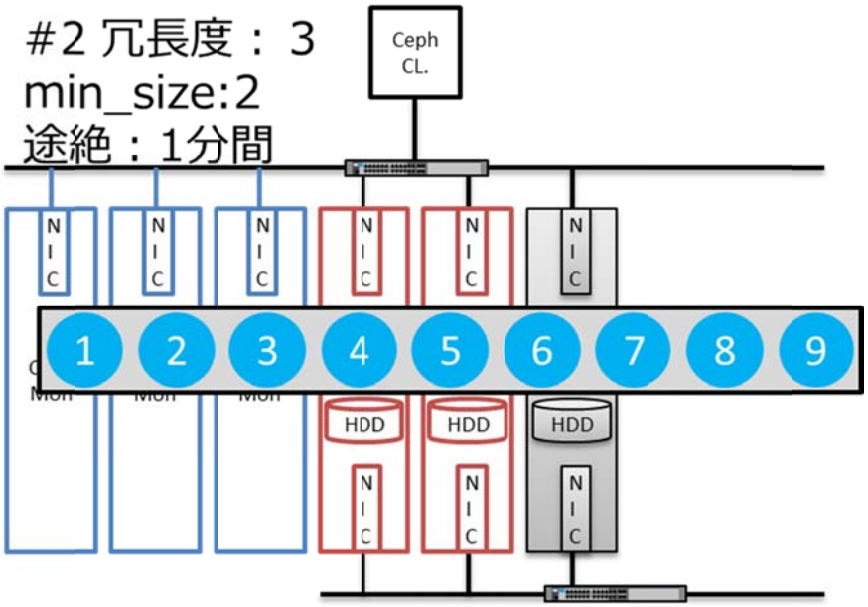
5.4. テスト結果

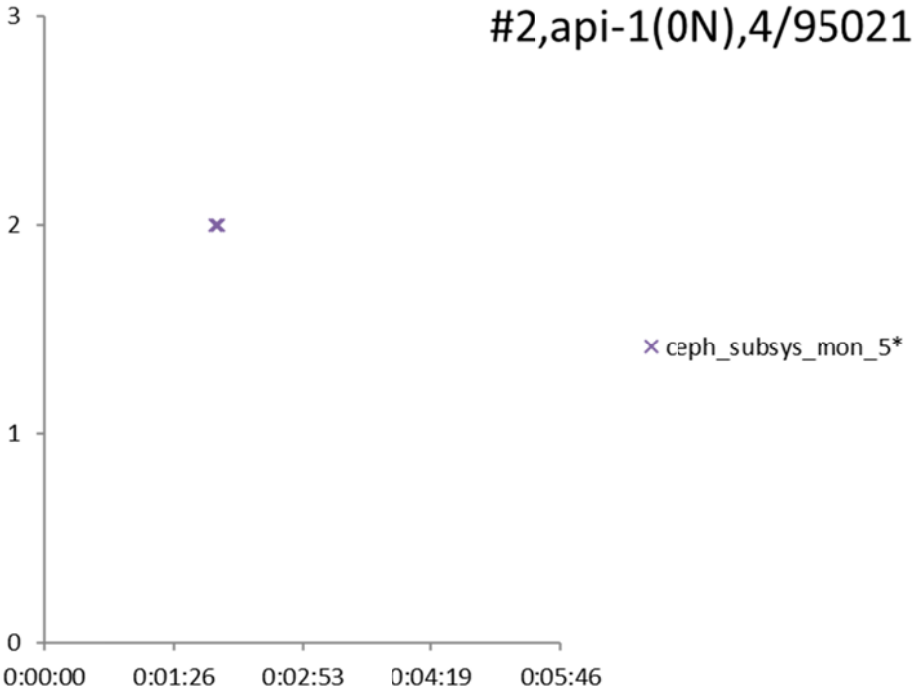
ここからは OpenStack/Ceph 異常系テスト (2) の結果のうち、ハードウェア障害のパターン毎に、以下のデータを 2 種類の図で紹介します。

- ハードウェア故障箇所 (システム構成図)
  ハードウェア故障箇所はシステム構成図中、灰色にマークされている箇所です。
  欄外には障害パターンの番号と必要に応じてデータ冗長度や min_size、ハードウェア障害が継続した時間の長さ等の補足情報を記載します。
- 9 個のオペレーションへの影響 (システム構成図の中央に記載)
  オペレーション番号 (api-1～api-9) 毎の観測された影響を青色、黄色、赤色の丸で表示します。
  青色は、ハードウェア障害の影響がなかったことを表します。
  黄色は、ハードウェア障害を取り除いた後で、当該オペレーションのリトライ等が必要だったことを表します。
  赤色は、ハードウェア障害を取り除いた後で、OpenStack 環境の不整合等を修復するためのオペレーションが必要だったことを表します。
  なお、観測された影響 (影響なしを含む) ついては以下の点に注意してください。
  - 現象の再現性は確認されていません。
  - 現象の原因が、発生させたハードウェア故障であることは確認されていません。
- 特徴ログの種類と行数、時間分布 (プロット図)
  9 個のオペレーションのうち、原則として api-1 のオペレーションのテスト項目の実行時のログから抽出された特徴ログの時間分布を示すグラフです。
  横軸を時間、縦軸をログ出力件数としてサブシステムおよびログレベル毎の特徴ログ件数をプロットします。
  ラベルは「サブシステム名_ログレベル」の形式です。
  欄外の表示は、「障害パターン番号、オペレーション番号 (特徴ログの種類) 、特徴ログ行数/全体ログ行数」です。
  特徴ログの種類は ON ログか NM ログかのいずれかです。
  api-1 のオペレーションでメモリ上のログ領域不足によりメモリログの上書きが発生している場合はログ取得開始時点からの冒頭部分のログが失われている可能性があるため、この場合は api-1 以外のオペレーションでログの冒頭部分が取得できているオペレーションを選択して特徴ログを抽出します。

以降、障害パターン毎にシステム構成図とプロット図を示します。また、9 個のオペレーションへの影響 (システム構成図の中央に記載) で黄色および赤色の箇所があるものについては、それぞれに対応する、観測された現象とハードウェア障害を取り除いた後で行った対処について列挙します。

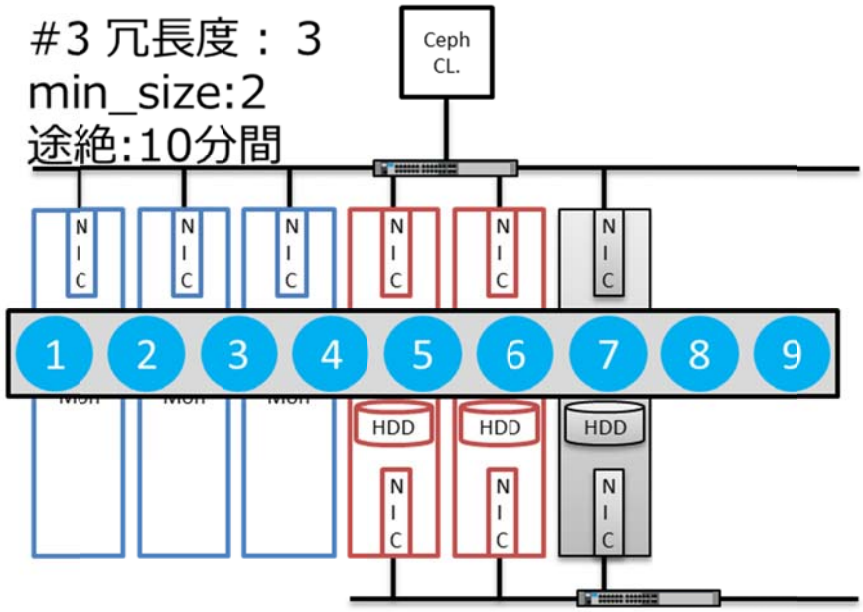
#2 OSD ネットワーク途絶

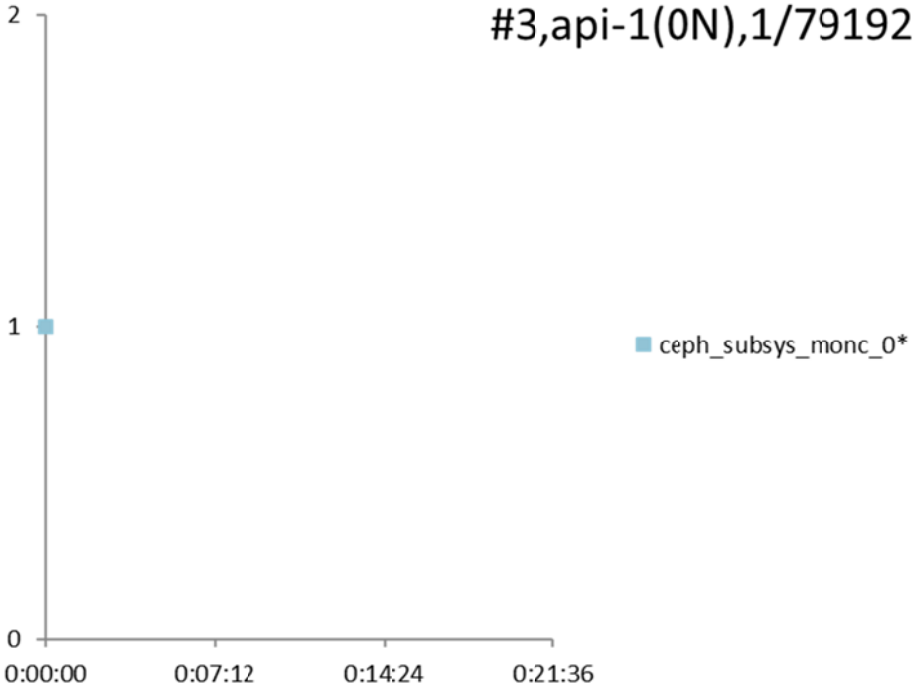




現象と対処：なし

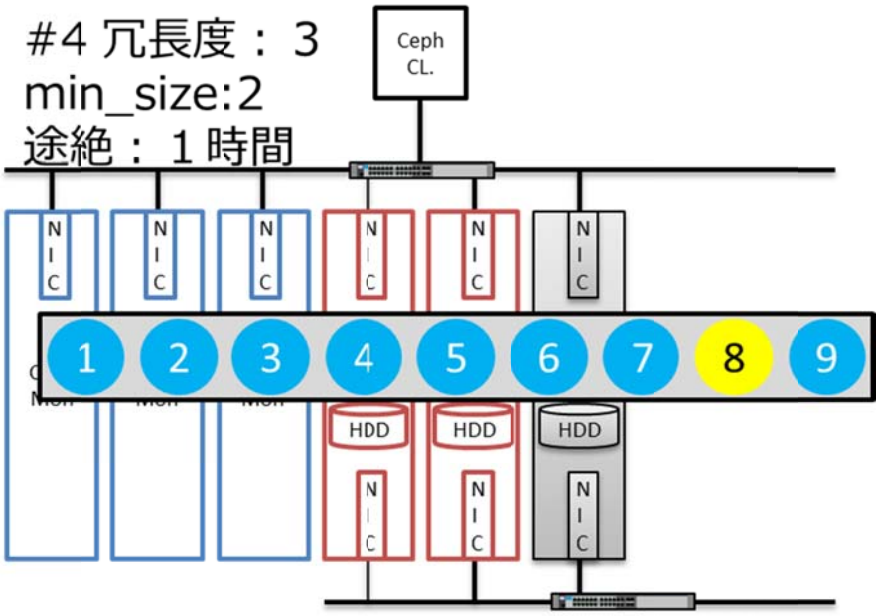
#3 OSD ネットワーク途絶

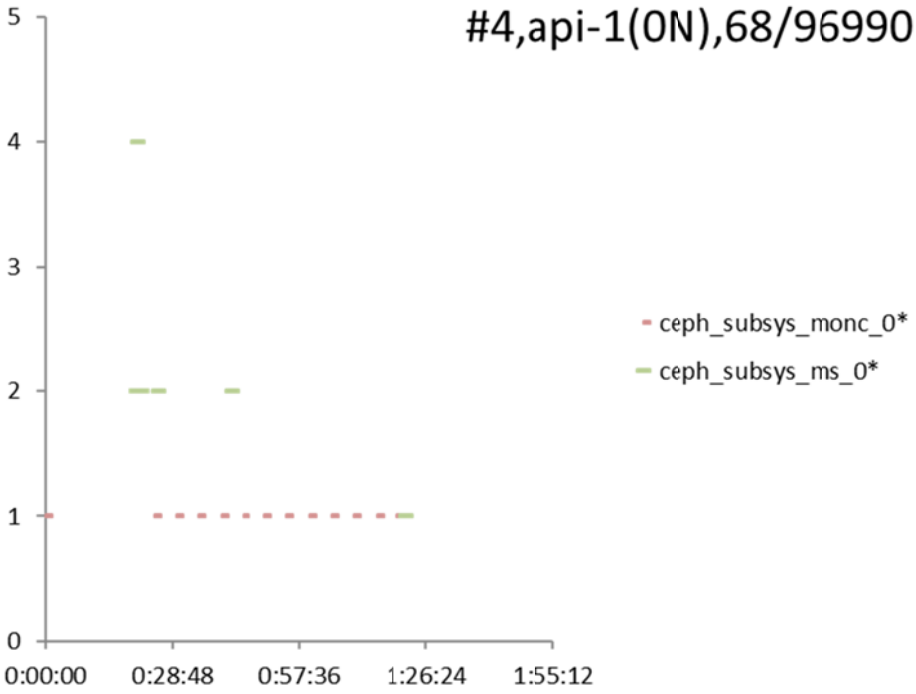




現象と対処：なし

#4 OSD ネットワーク途絶





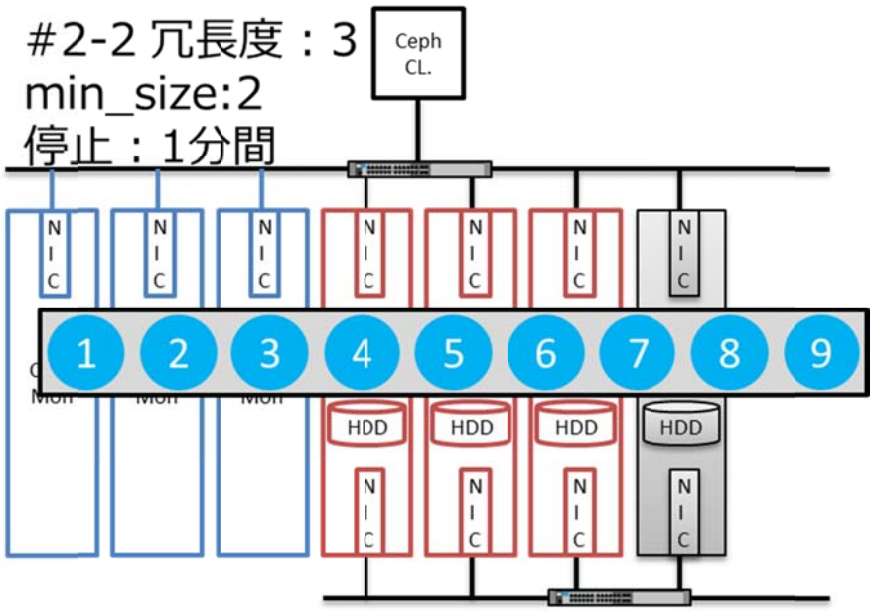
現象と対処：

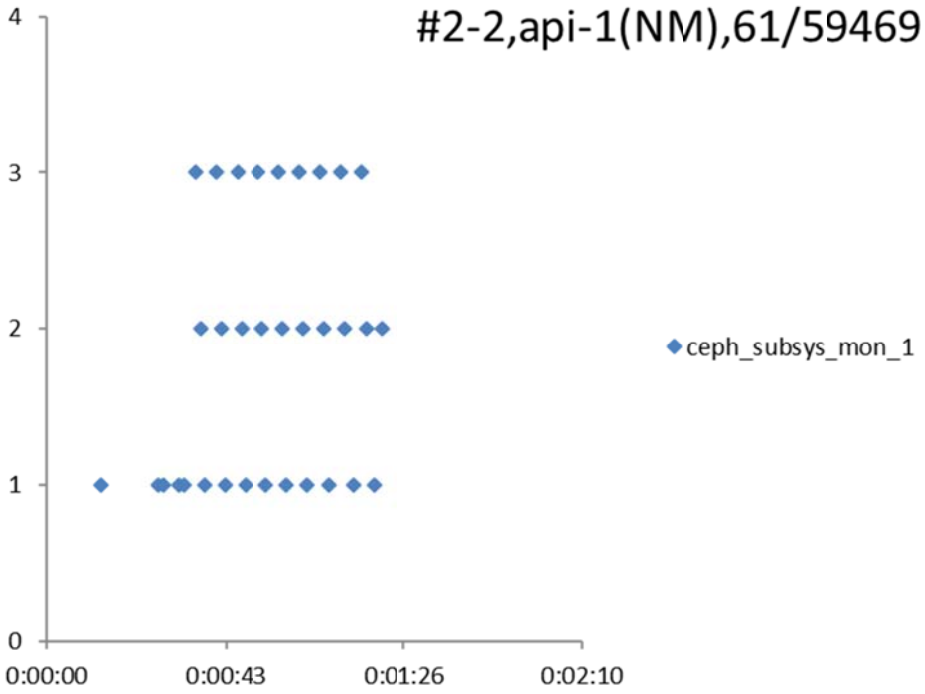
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を再削除

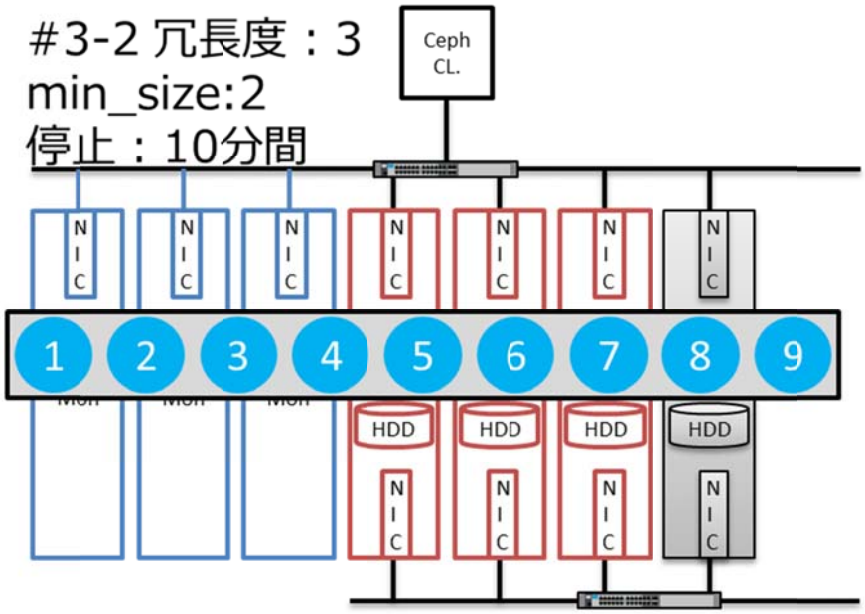
#2-2 OSD 停止

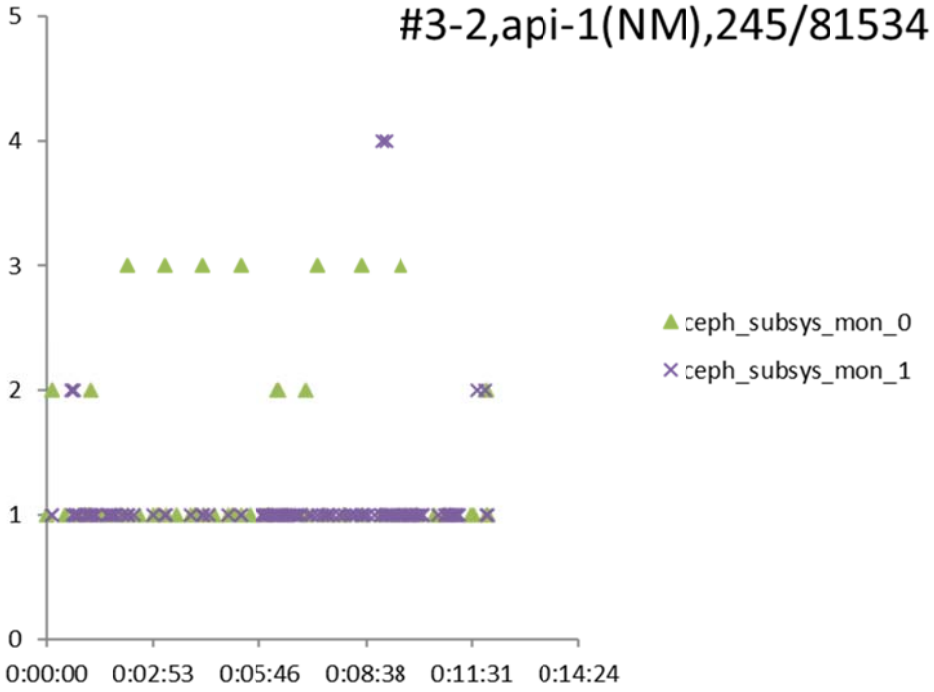




現象と対処：なし

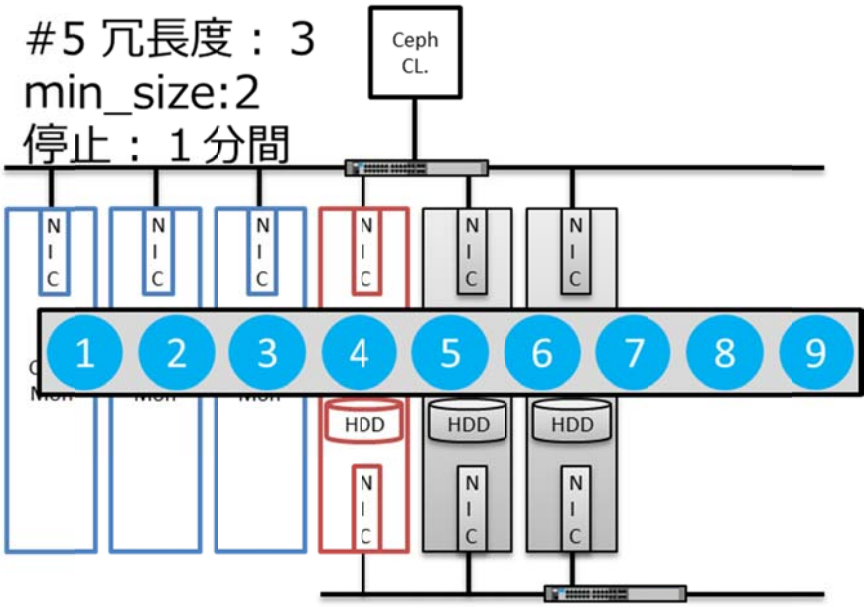
#3-2 OSD 停止

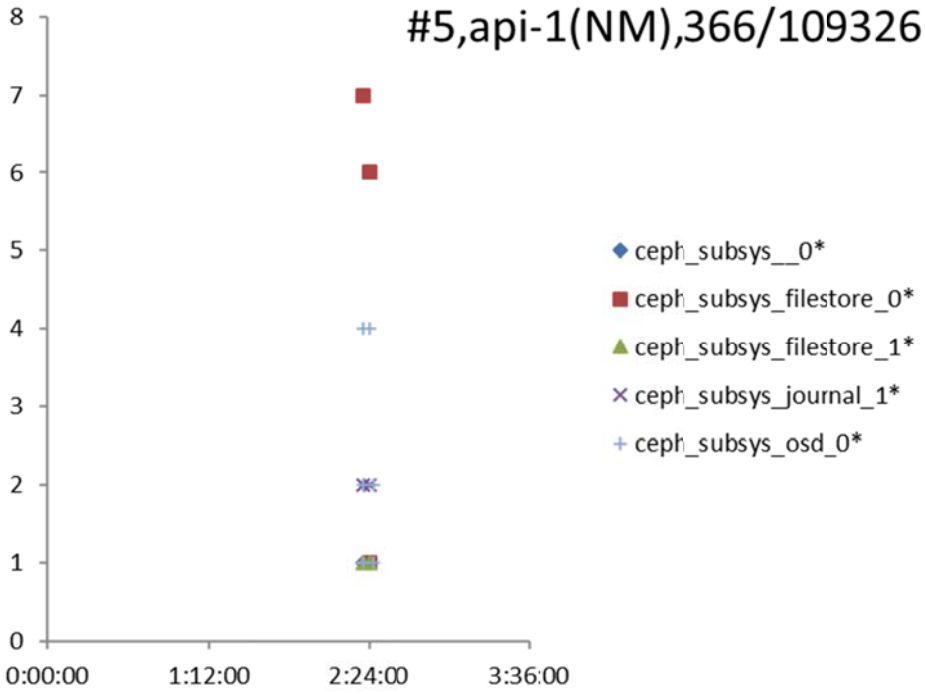




現象と対処：なし

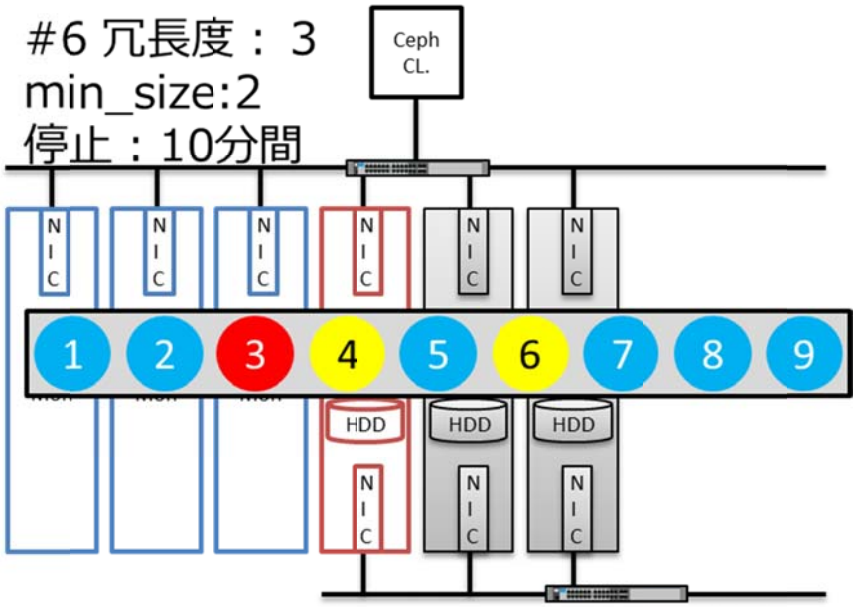
#5 OSD 停止

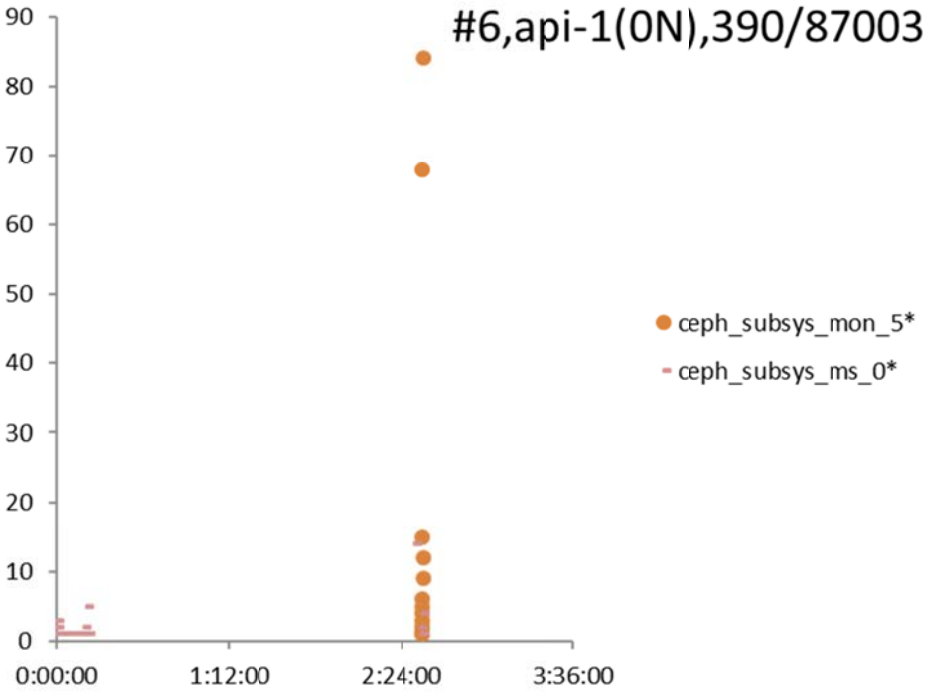




現象と対処：なし

#6 OSD 停止





現象と対処：

オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: image_pending_upload)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

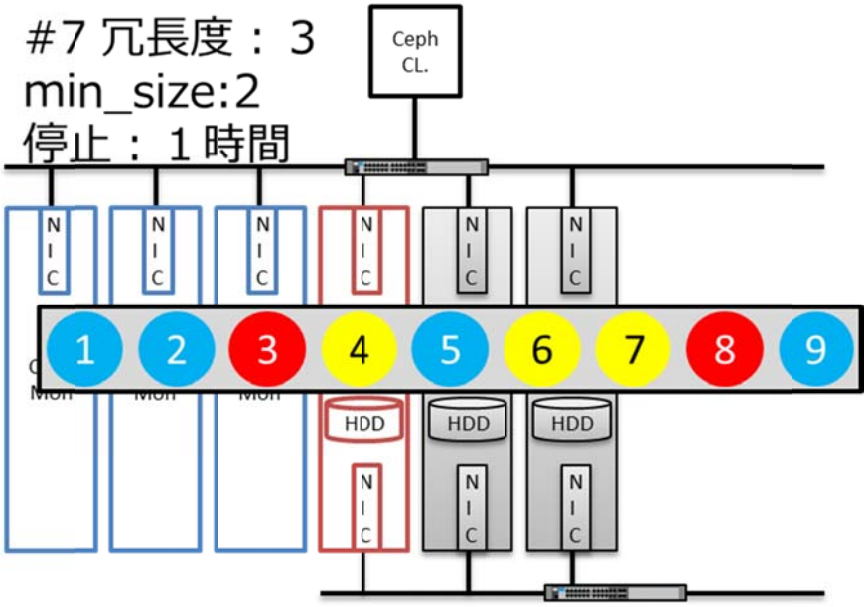
オペレーション: api-6

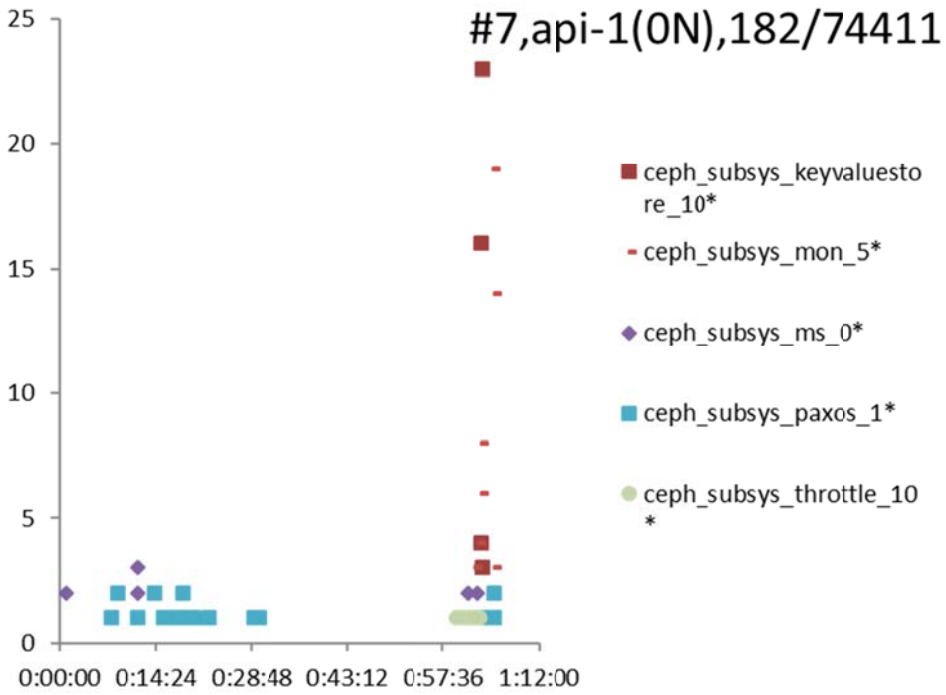
現象: スナップショット-C からインスタンス-D が起動しない (Status: BUILD, Task State: spawning)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

#7 OSD 停止





現象と対処：

オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: -)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

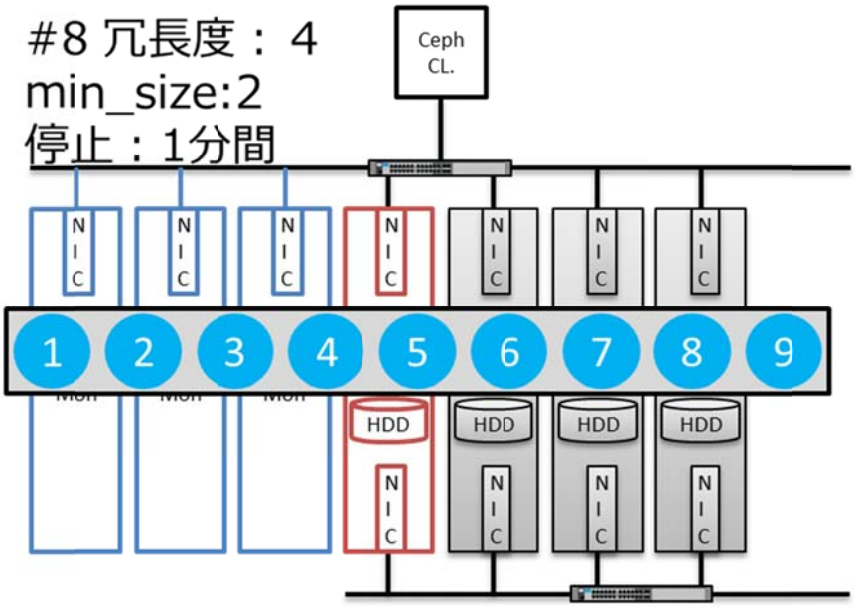
対処: インスタンス-D を削除

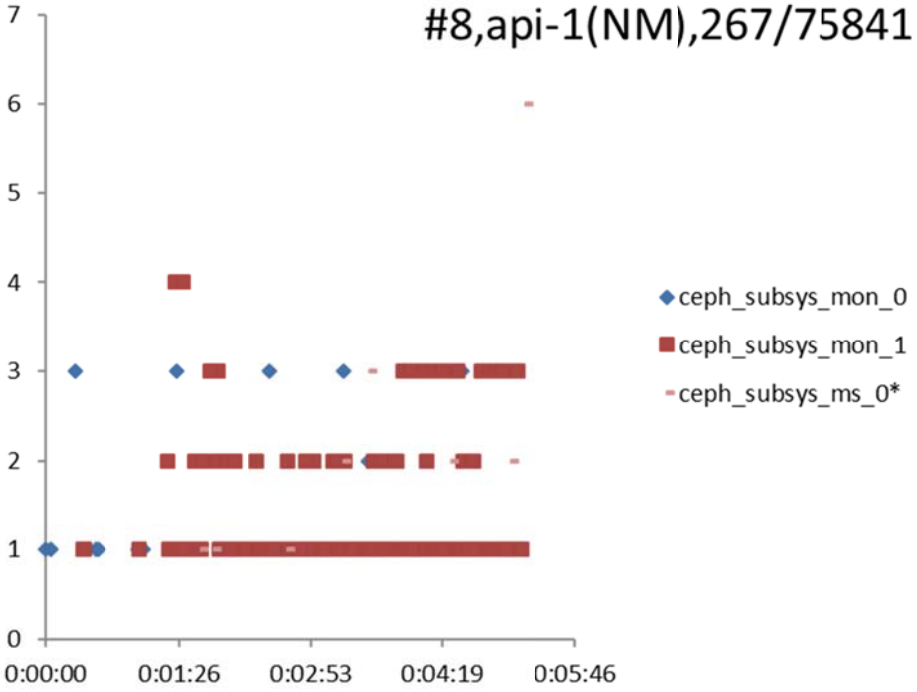
オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

対処: "glance" データベースの "images" テーブル上の "name" の該当するエントリの "deleted" の値を 0 から 1 に変更する。

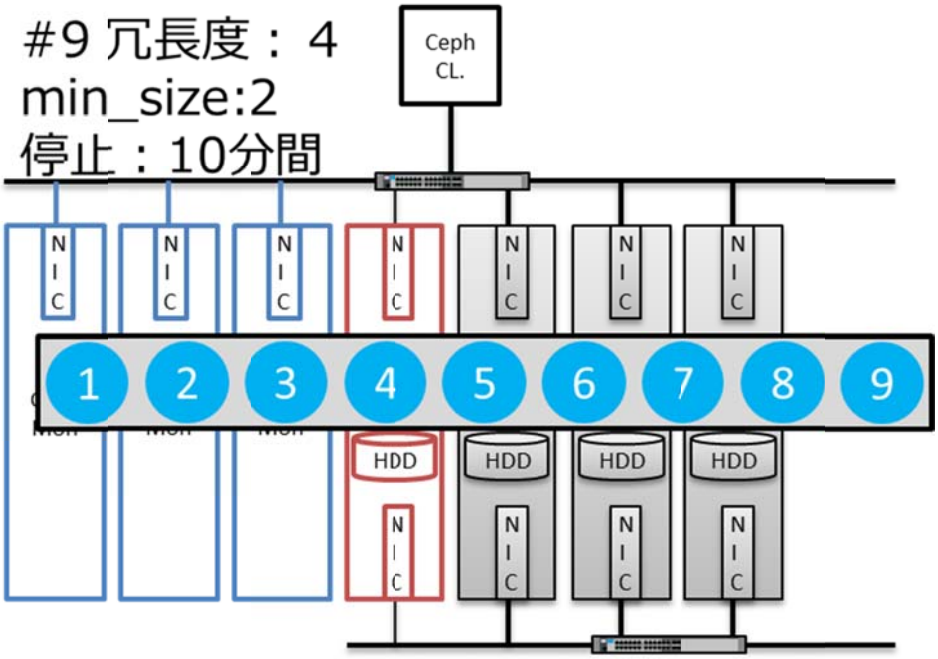
#8 OSD 停止

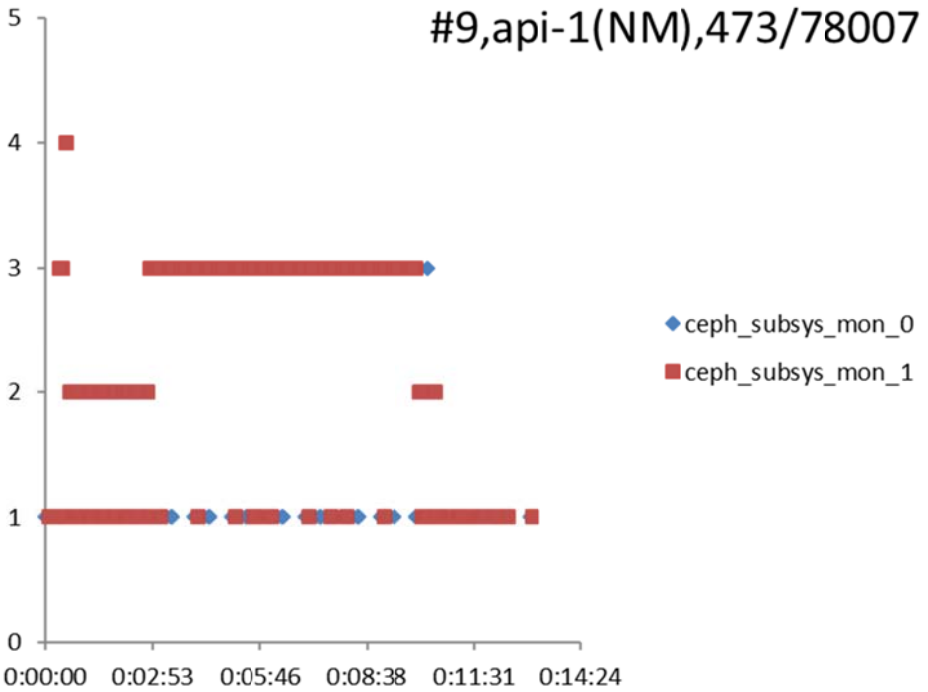




現象と対処：なし

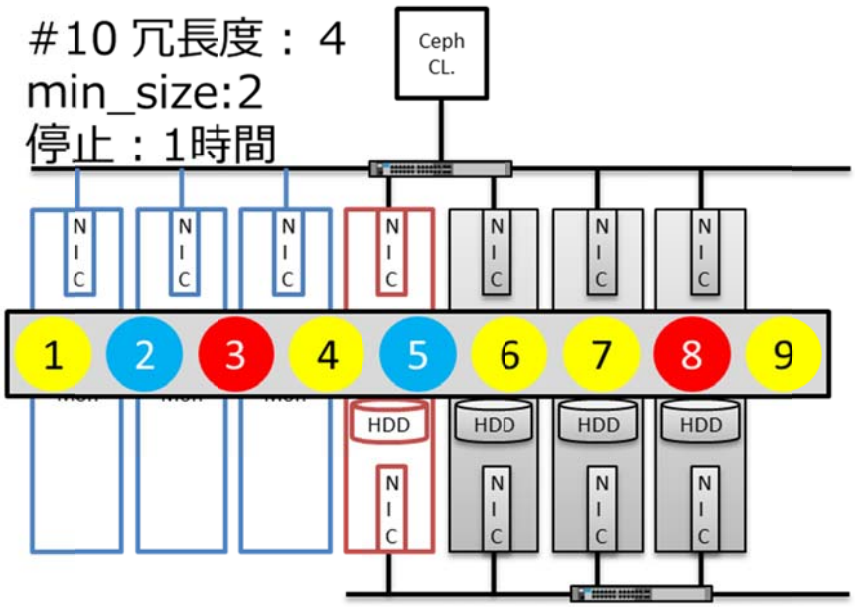
#9 OSD 停止

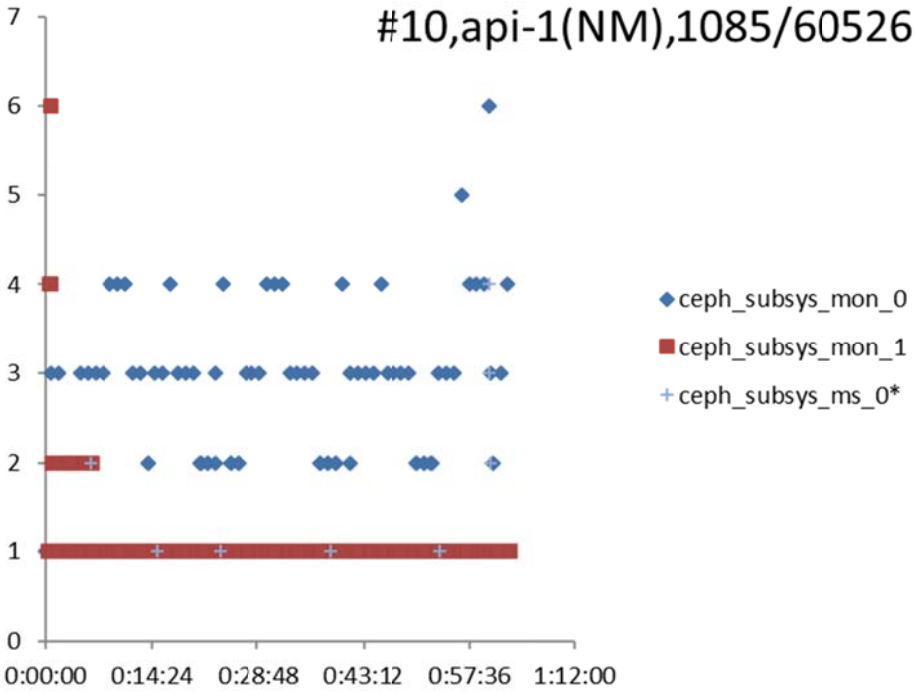




現象と対処：なし

#10 OSD 停止





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:ERROR, Task State: -)

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: -)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

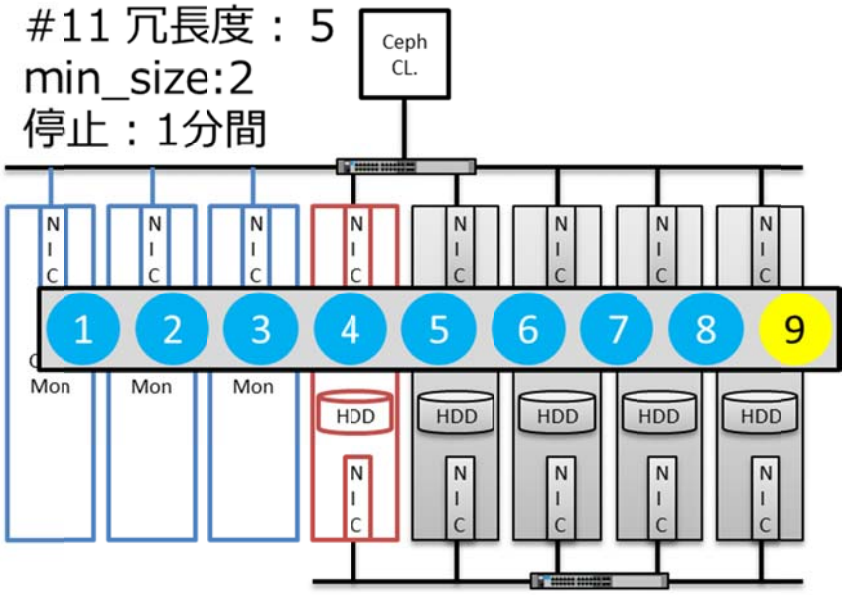
対処: "glance" データベースの "images" テーブル上の "name" の該当するエントリの "deleted" の値を 0 から 1 に変更する。

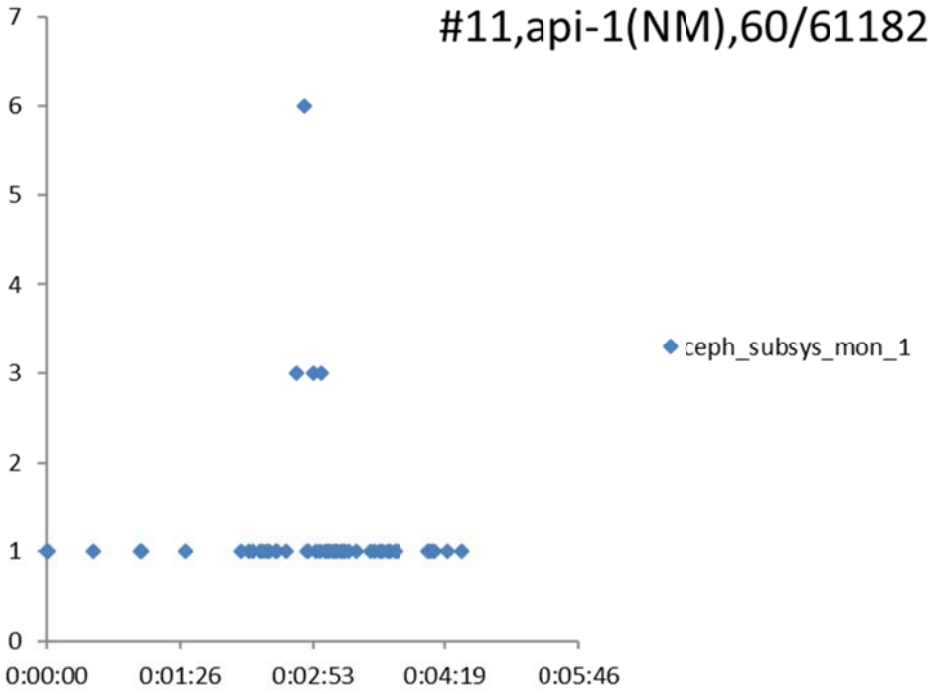
オペレーション: api-9

現象: ボリューム-B が削除されない (Status: available)

対処: ボリューム-B を削除

#11 OSD 停止





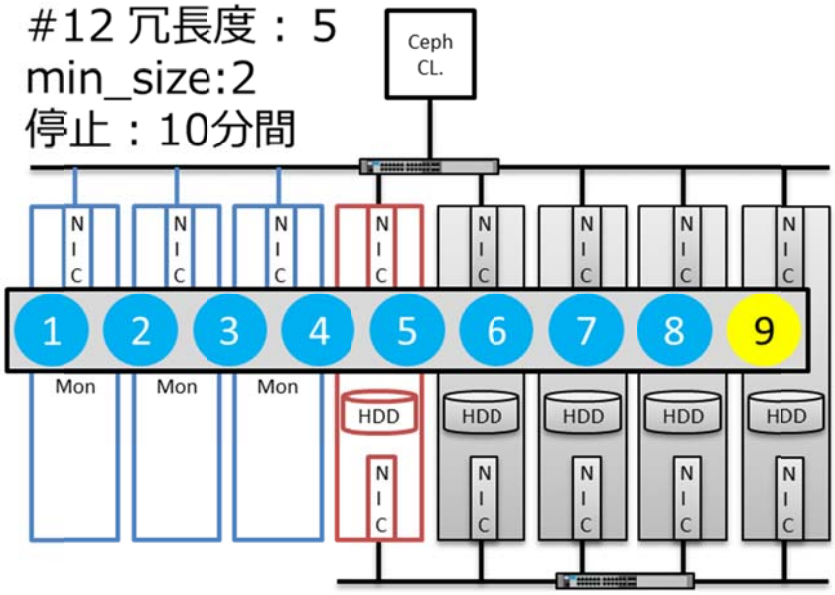
現象と対処：

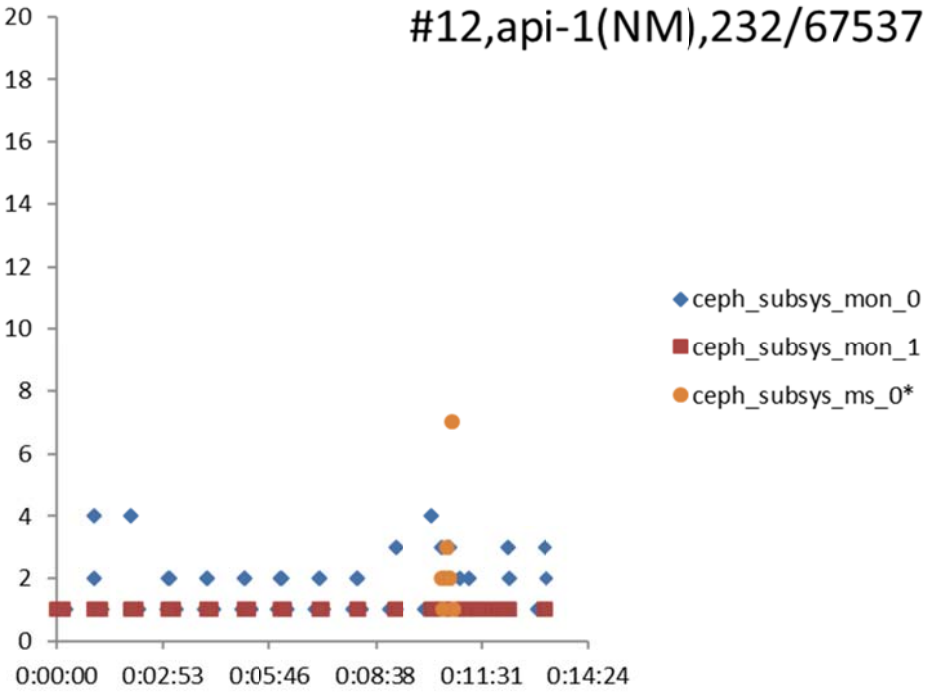
オペレーション： api-9

現象： ボリューム-B が削除されない (Status: available)

対処： ボリューム-B を削除

#12 OSD 停止





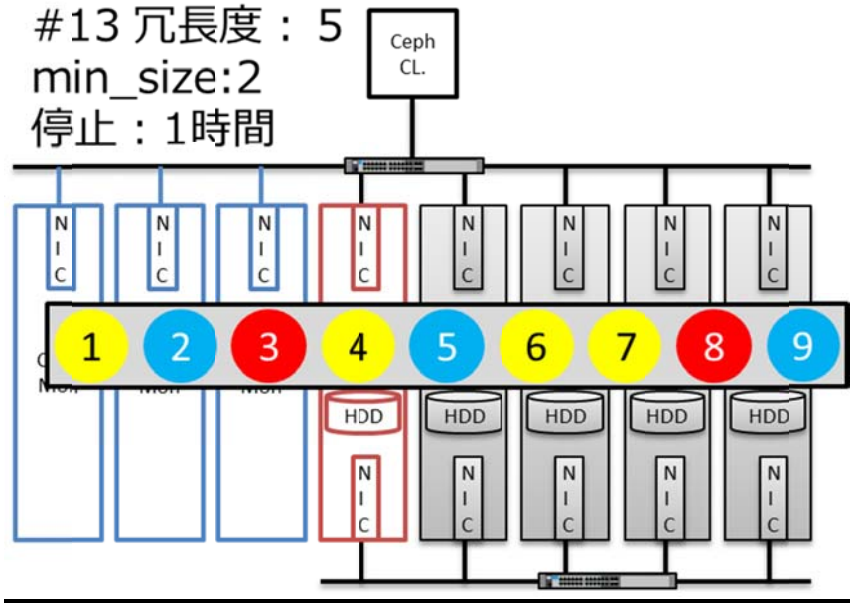
現象と対処：

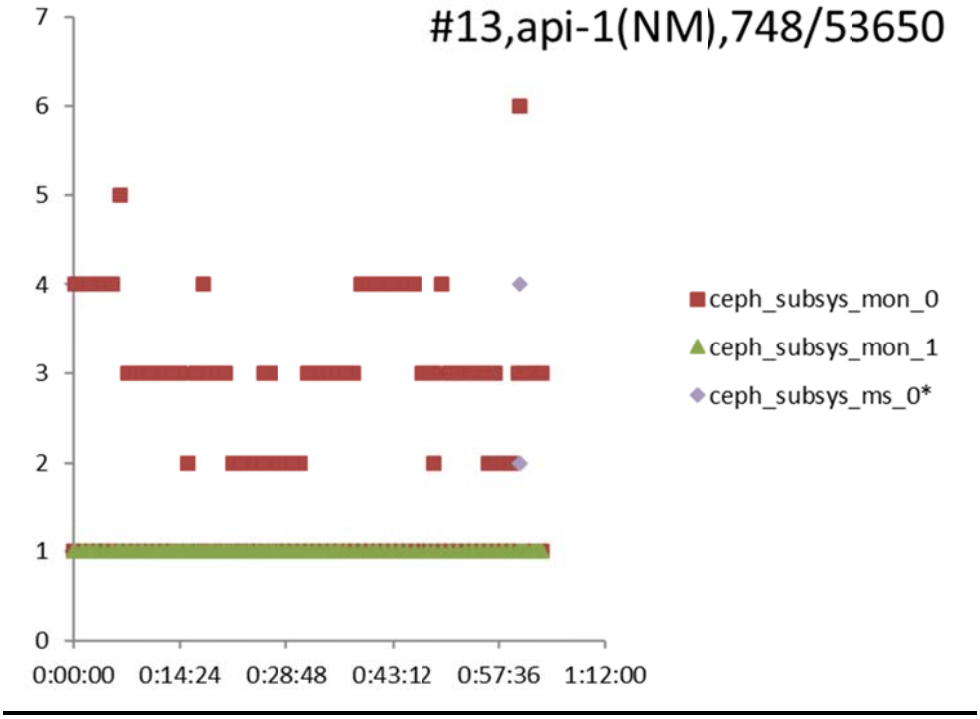
オペレーション： api-9

現象： ボリューム-B が削除されない (Status: available)

対処： ボリューム-B を削除

#13 OSD 停止





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:ERROR, Task State: spawning)

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象:インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: -)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象:スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

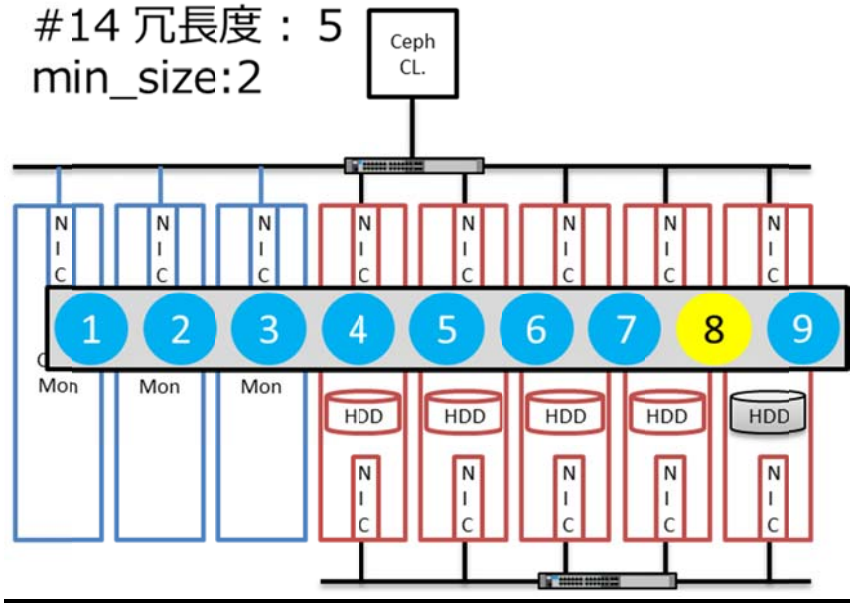
対処: インスタンス-D を削除

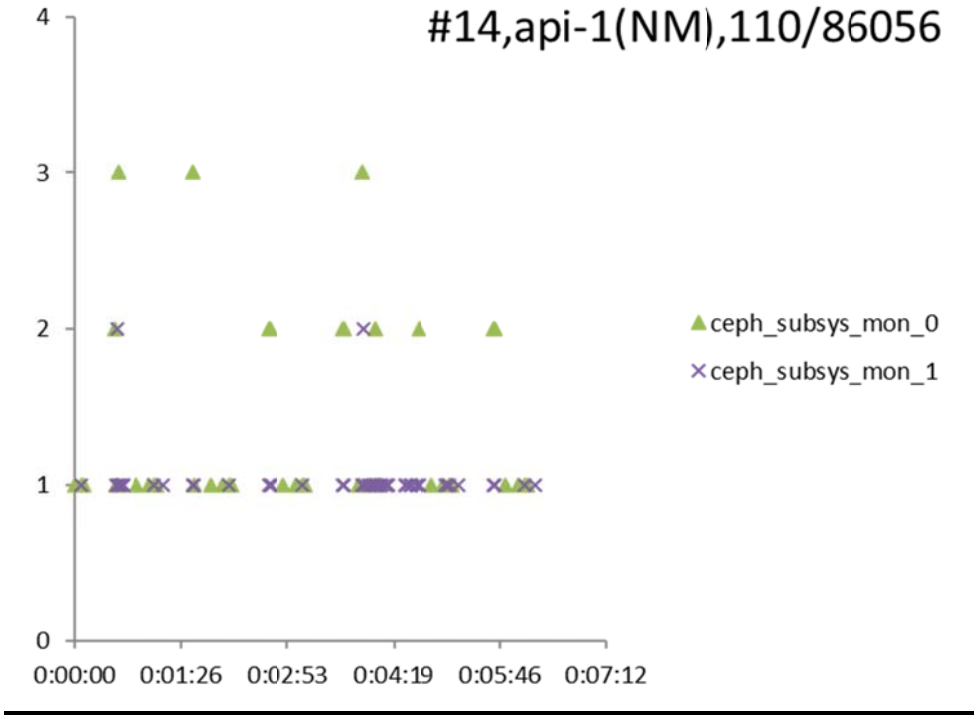
オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

対処: "glance" データベースの "images" テーブル上の "name" の該当するエントリの "deleted" の値を 0 から 1 に変更する。

#14 ディスクフル





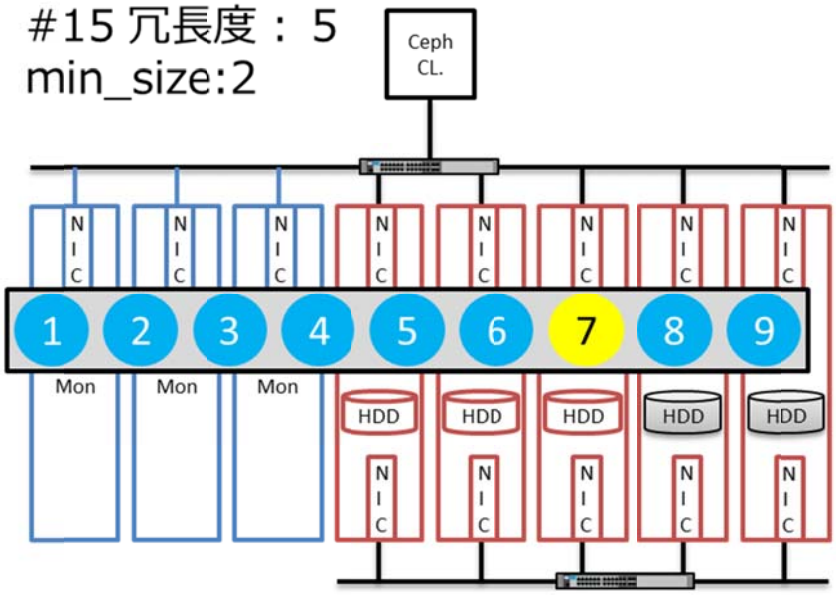
現象と対処：

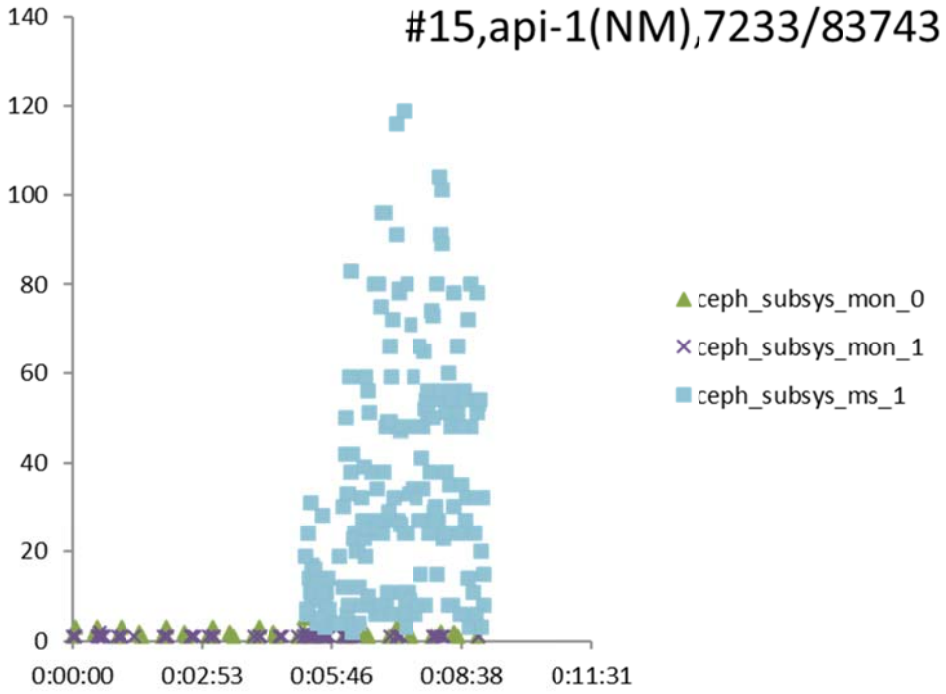
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#15 ディスクフル





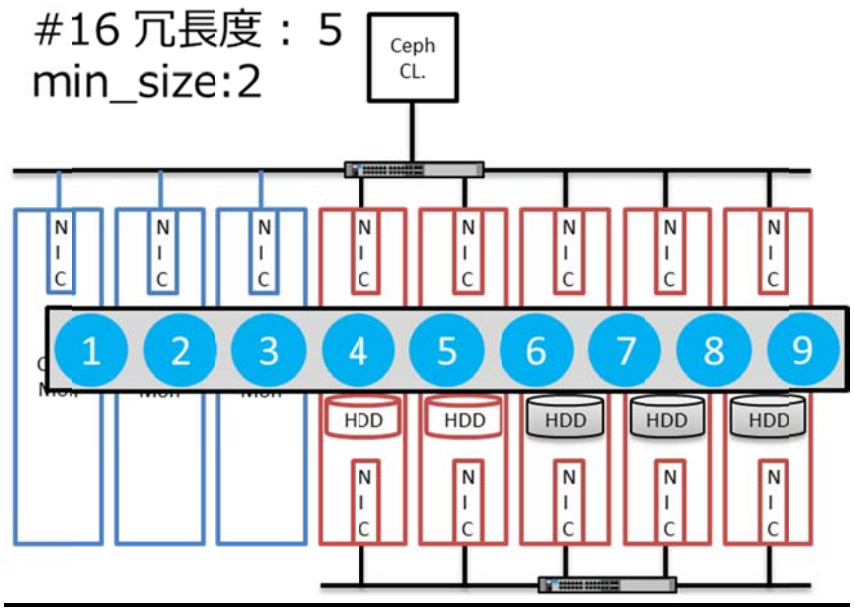
現象と対処：

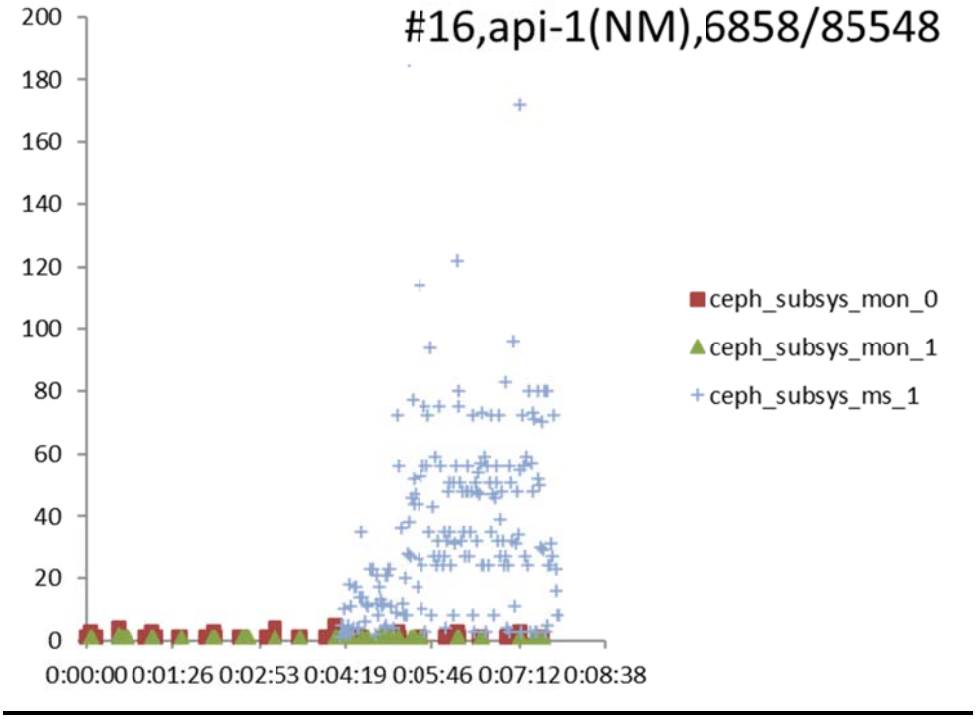
オペレーション： api-7

現象： インスタンス-D が削除されない (Status: ERROR, Task State: -)

対処： インスタンス-D を削除

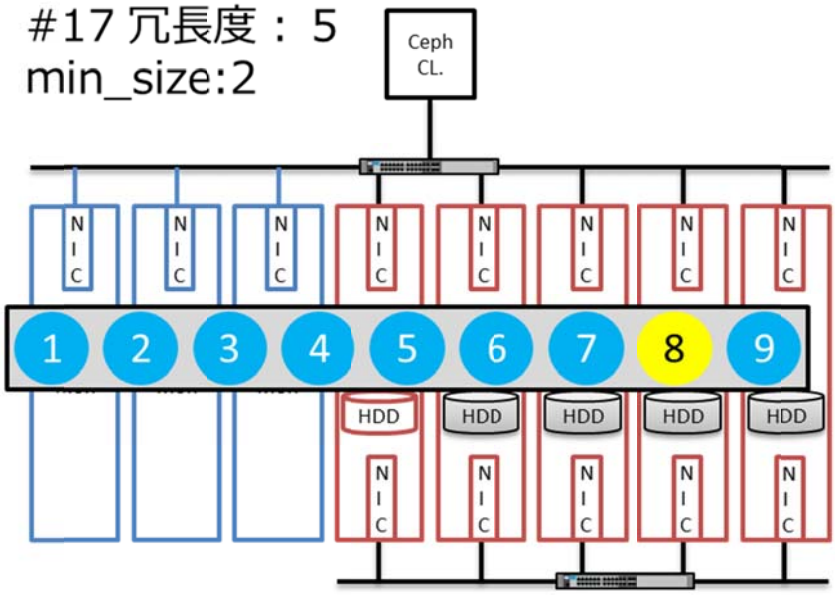
#16 ディスクフル

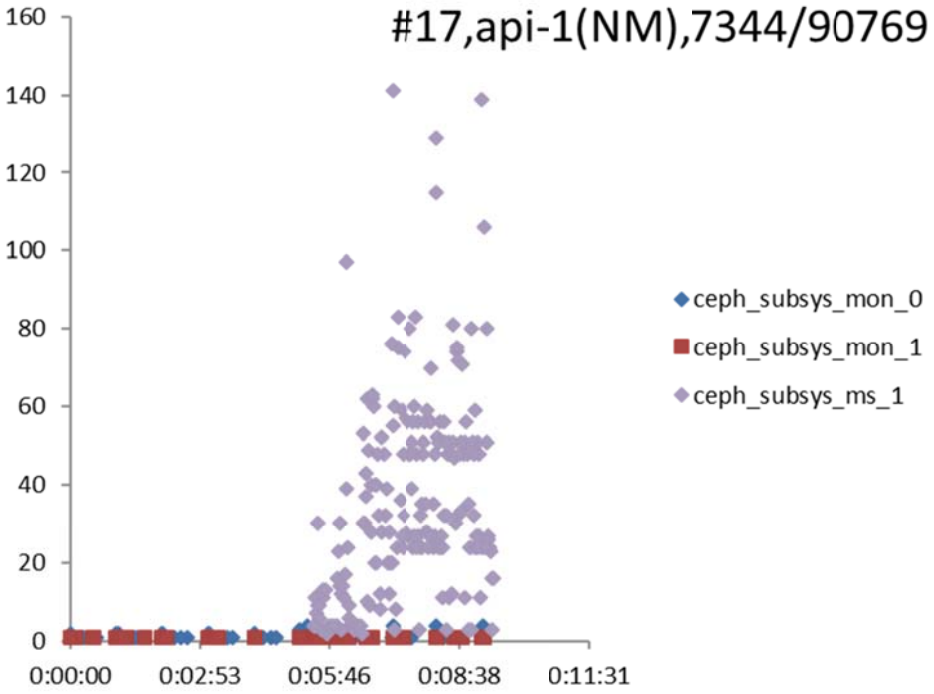




現象と対処：なし

#17 ディスクフル





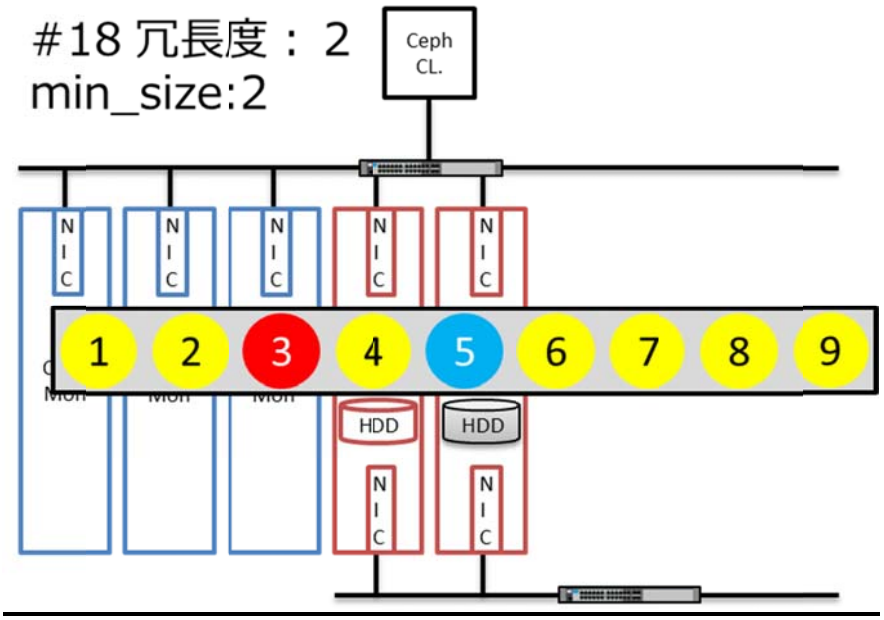
現象と対処：

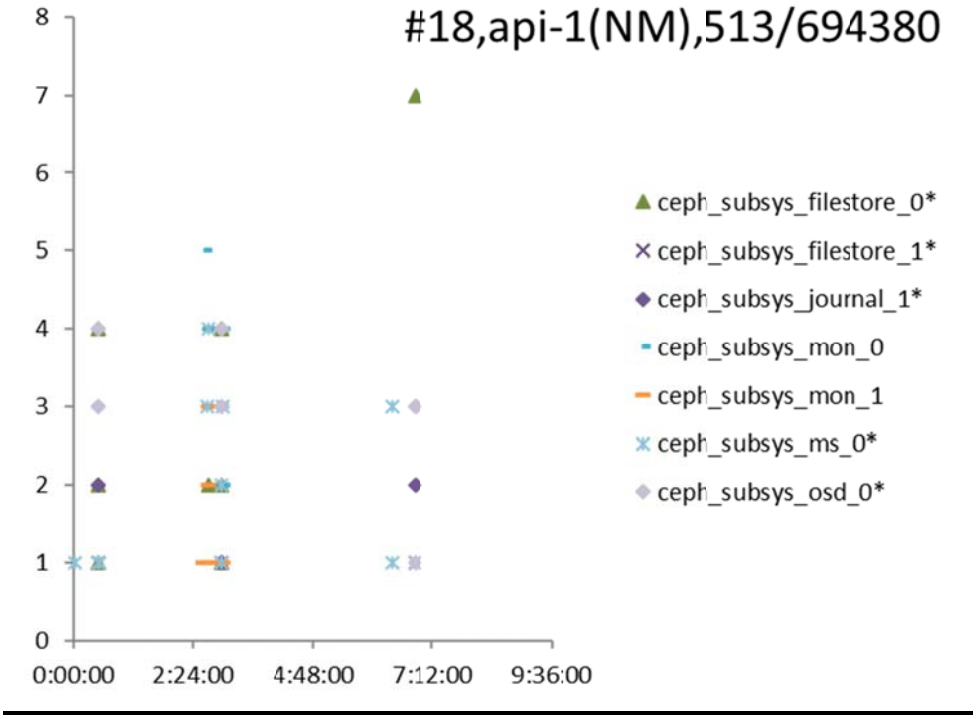
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#18 ディスク故障





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:BUILD, Task State: spawning)

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-2

現象: ボリューム-B が作成されない (Status: creating)

対処:

1) ボリューム-B を削除
2) ボリューム-B を再作成

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: image_pending_upload)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: BUILD, Task State: spawning)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ACTIVE, Task State: deleting)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

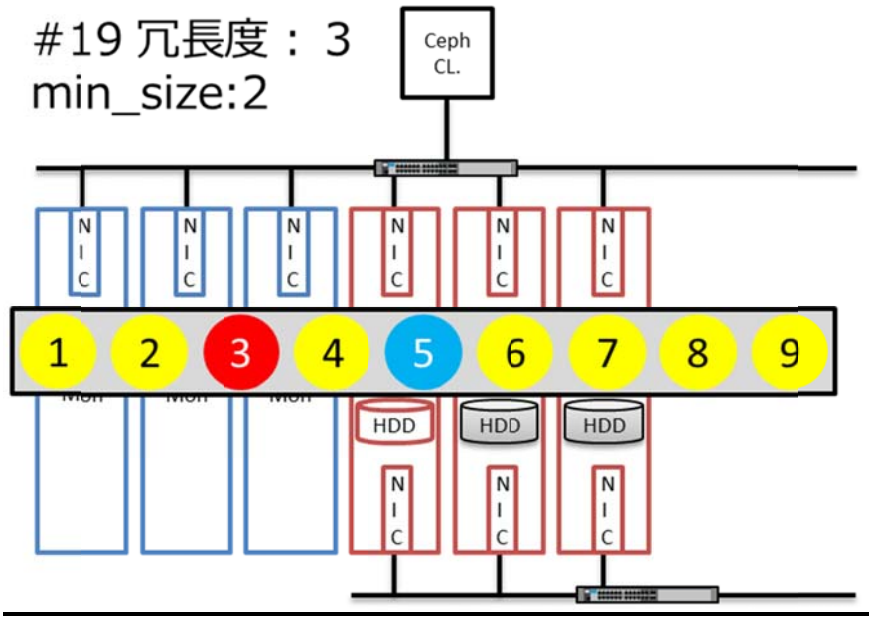
対処: スナップショット-C を削除

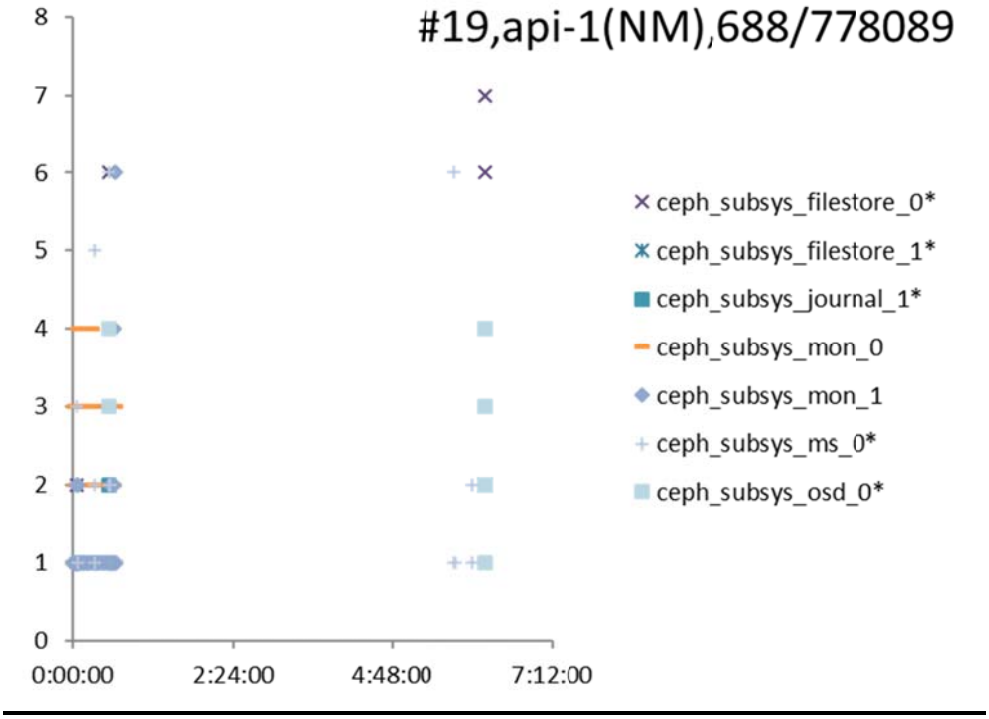
オペレーション: api-9

現象: ボリューム-B が削除されない (Status: deleting)

対処: ボリューム-B を削除

#19 ディスク故障





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:BUILD, Task State: spawning)

対処：

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-2

現象: ボリューム-B が作成されない (Status: creating)

対処:

1) ボリューム-B を削除
2) ボリューム-B を再作成

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: image_pending_upload)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: BUILD, Task State: spawning)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ACTIVE, Task State: deleting)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

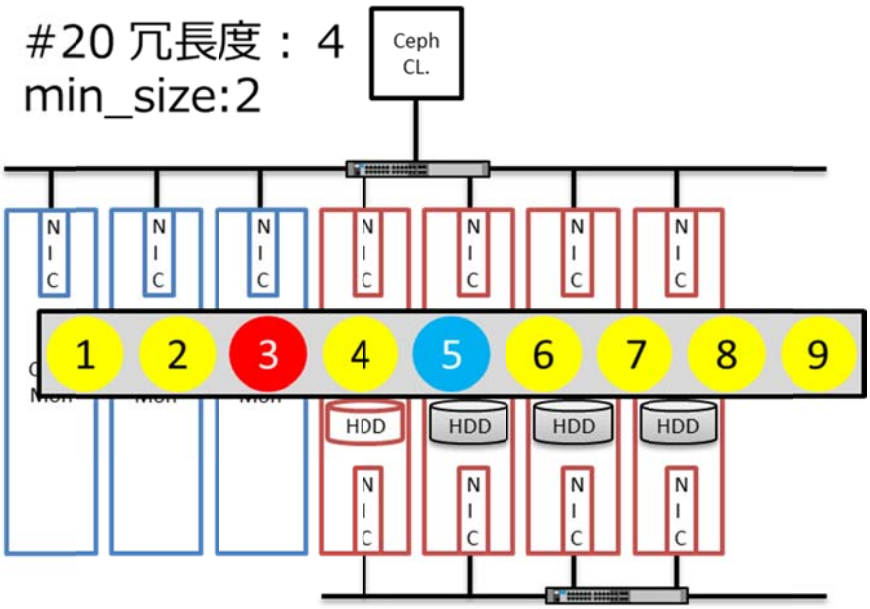
対処: スナップショット-C を削除

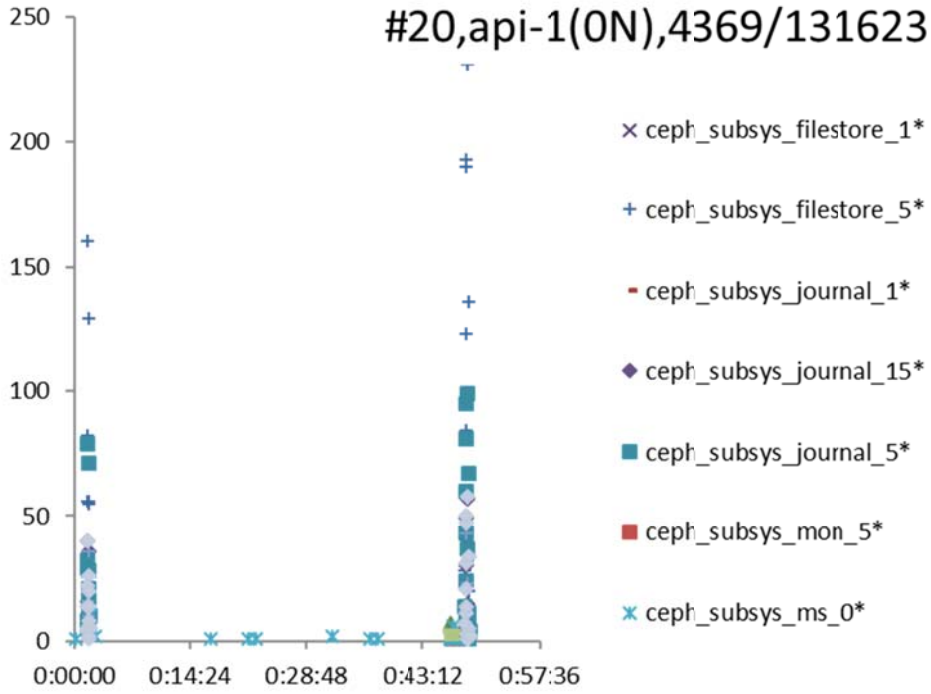
オペレーション: api-9

現象: ボリューム-B が削除されない (Status: deleting)

対処: ボリューム-B を削除

#20 ディスク故障





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:BUILD, Task State: spawning)

対処：

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-2

現象: ボリューム-B が作成されない (Status: creating)

対処:

1) ボリューム-B を削除
2) ボリューム-B を再作成

オペレーション: api-3

現象:ボリューム-B がインスタンス-A に接続しない (Status: attaching)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。

2) ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: image_pending_upload)

対処:

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: BUILD, Task State: spawning)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象:イ ンスタンス-D が削除されない (Status: ACTIVE, Task State: deleting)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

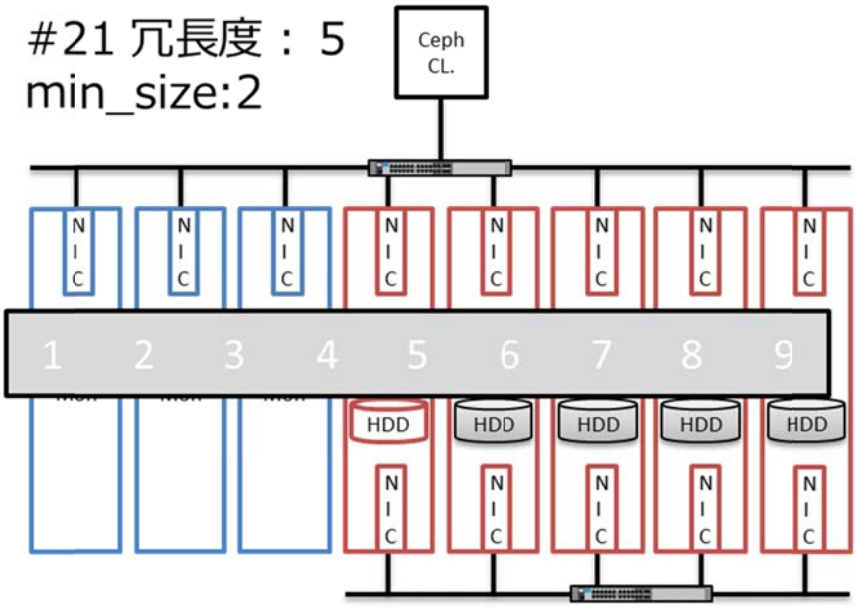
対処: スナップショット-C を削除

オペレーション: api-9

現象: ボリューム-B が削除されない (Status: deleting)

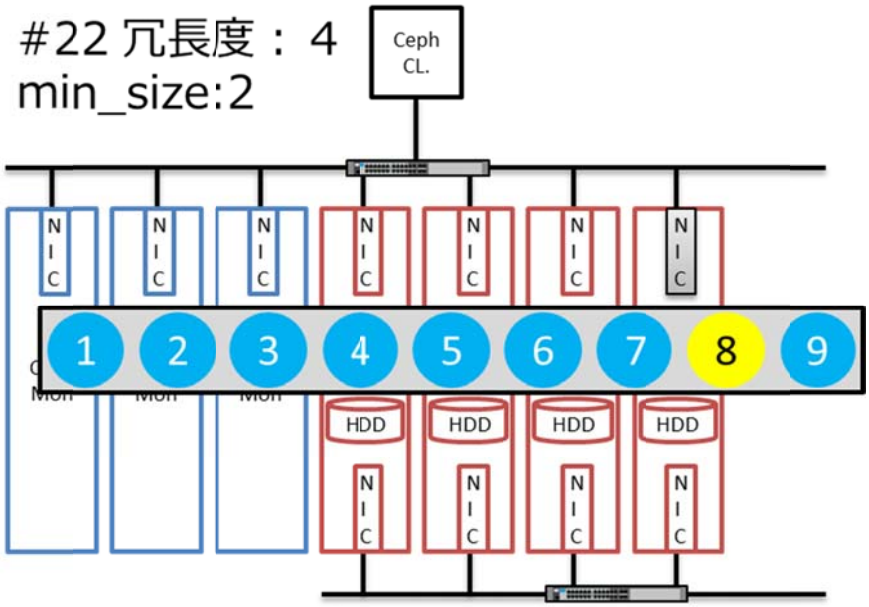
対処: ボリューム-B を削除

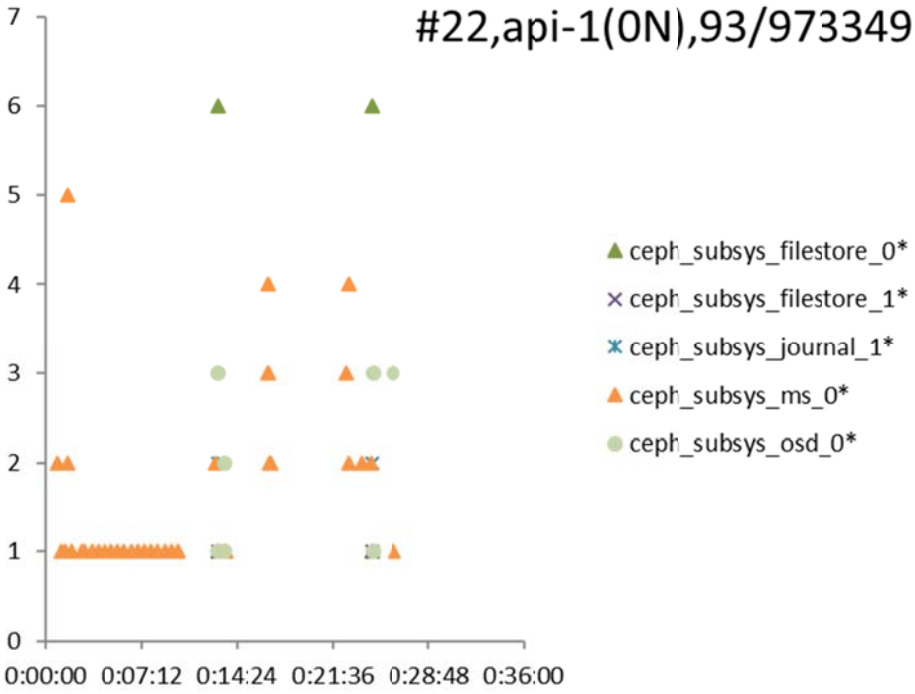
#21 ディスク故障



データ冗長度 5 の設定は組み合わせから除外しました。システム管理者の操作への影響という観点から、データ冗長度は 4 までの設定 (#20) で十分と考えたことが除外した主な理由です。

#22 public NIC 故障 (1)





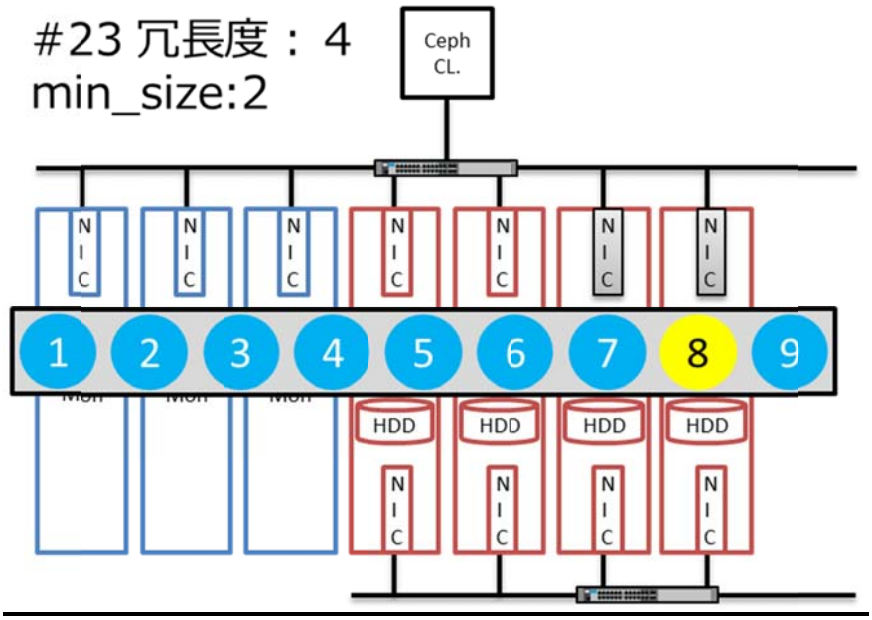
現象と対処：

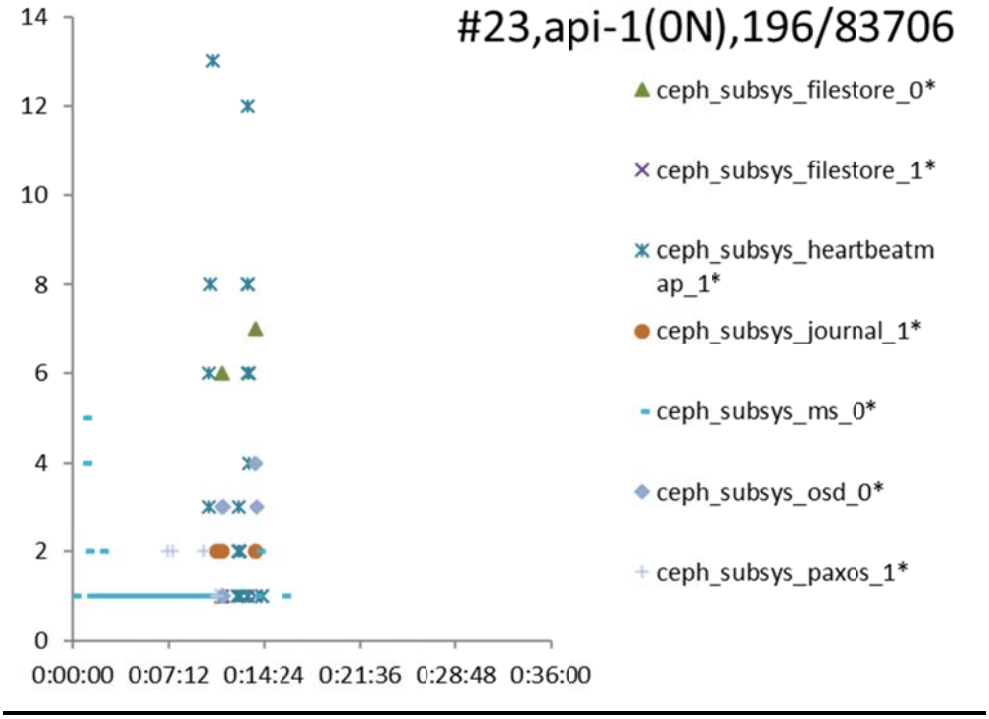
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#23 public NIC 故障 (1)





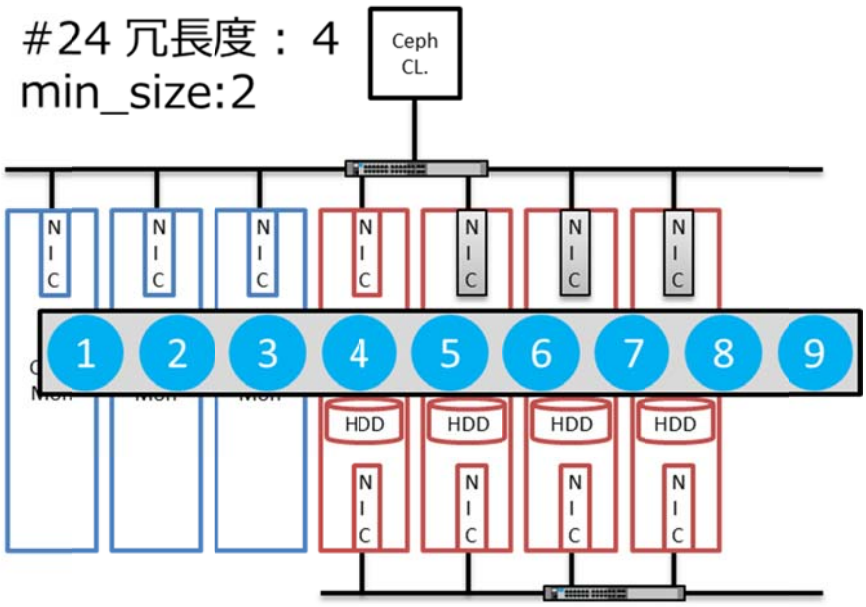
現象と対処：

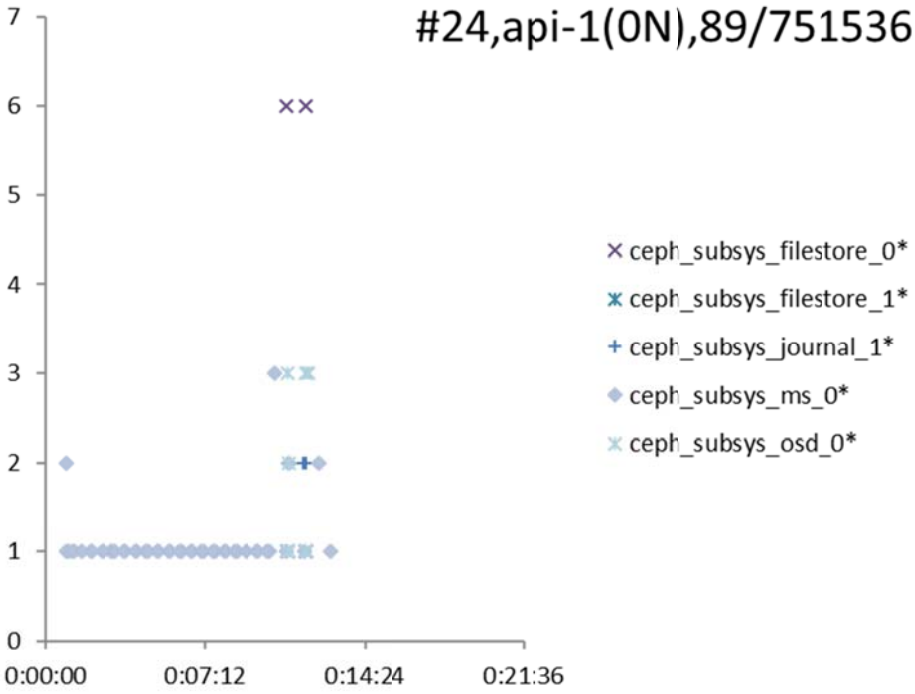
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

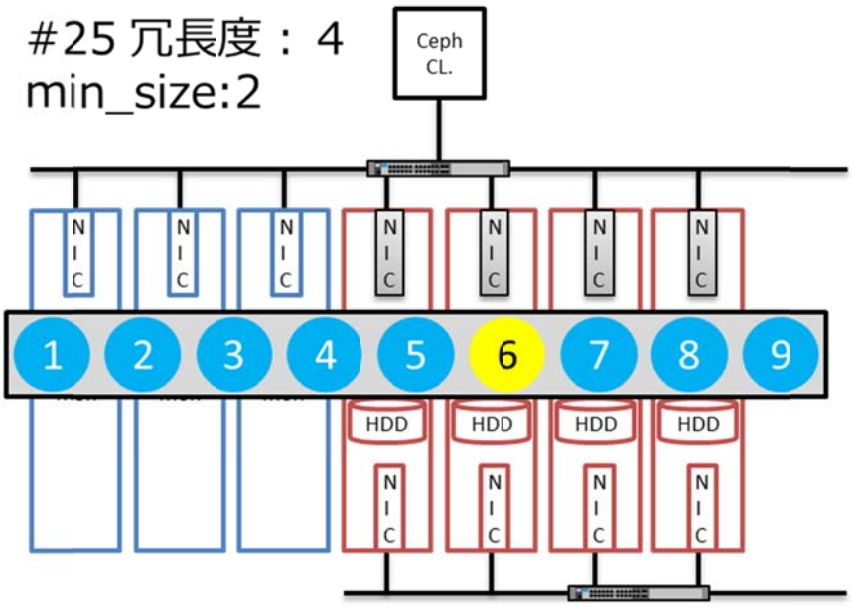
#24 public NIC 故障 (1)

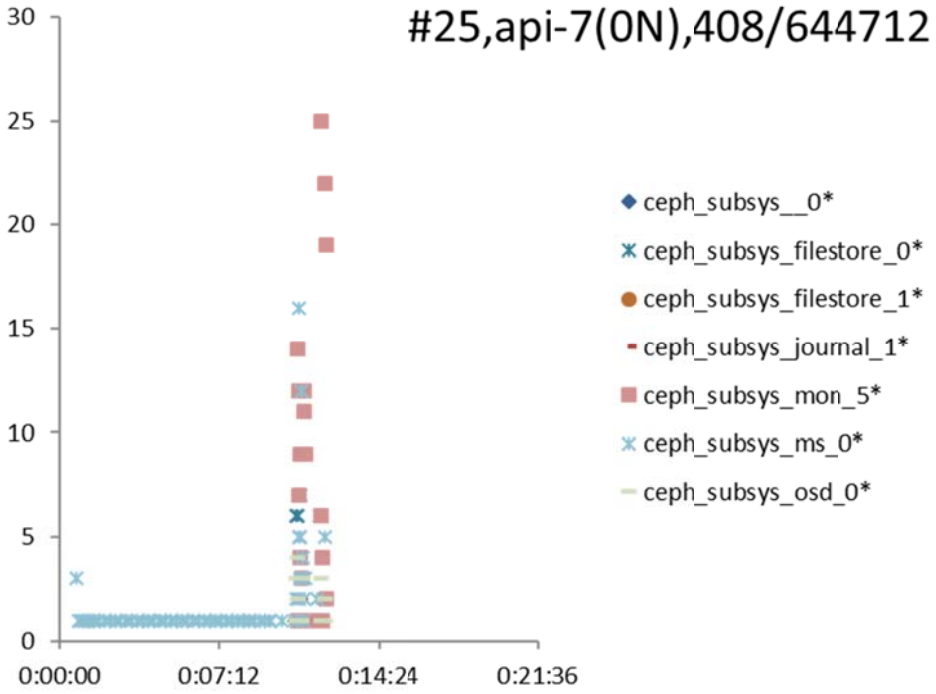




現象と対処：なし

#25 public NIC 故障 (1)





現象と対処:

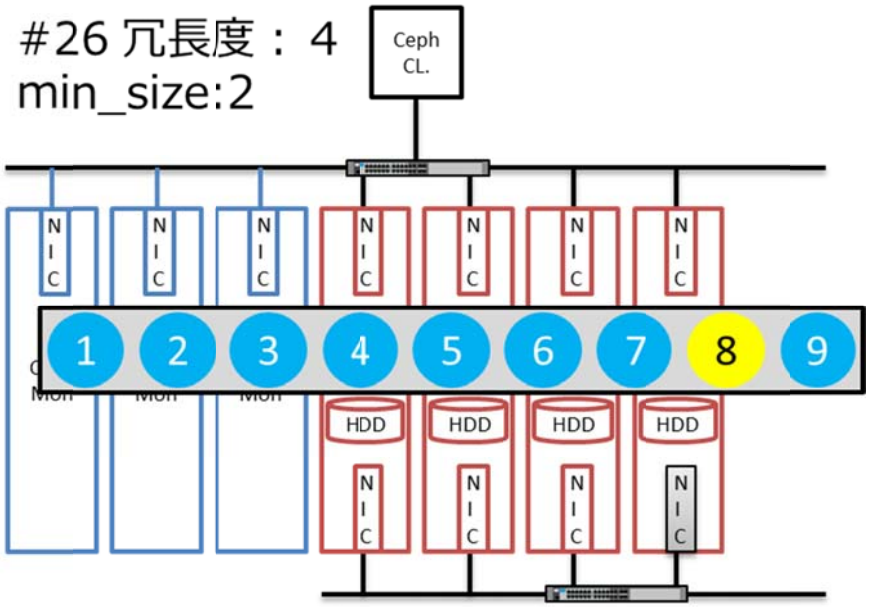
オペレーション: api-6

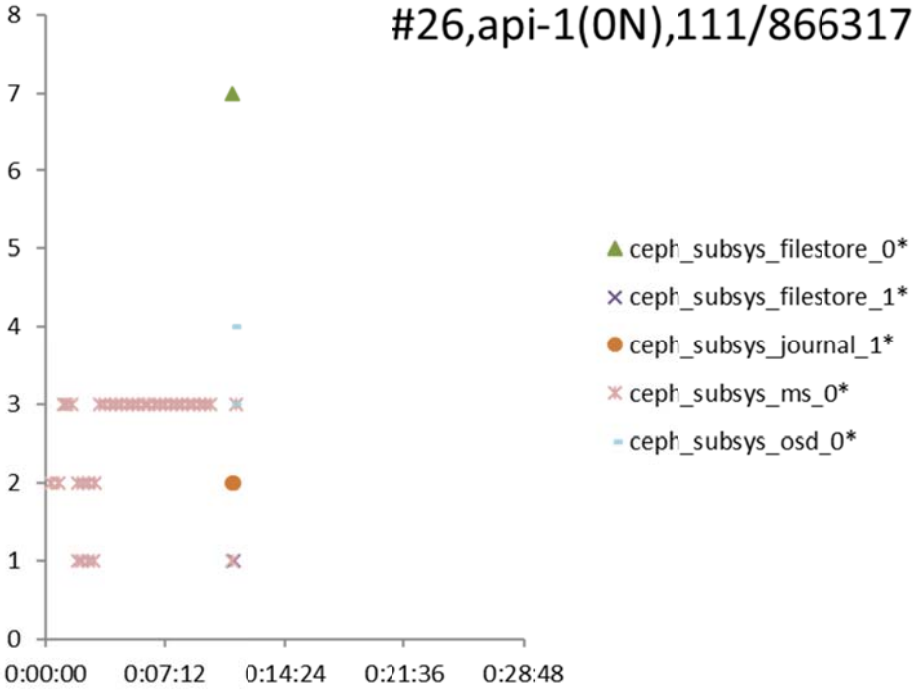
現象: スナップショット-C からインスタンス-D が起動しない (Status: BUILD, Task State: spawning)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

#26 cluster NIC 故障





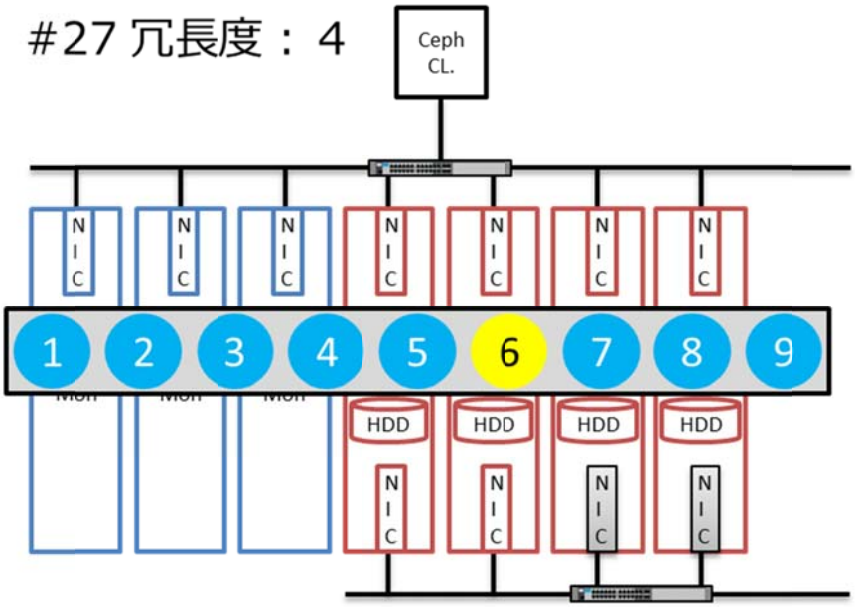
現象と対処：

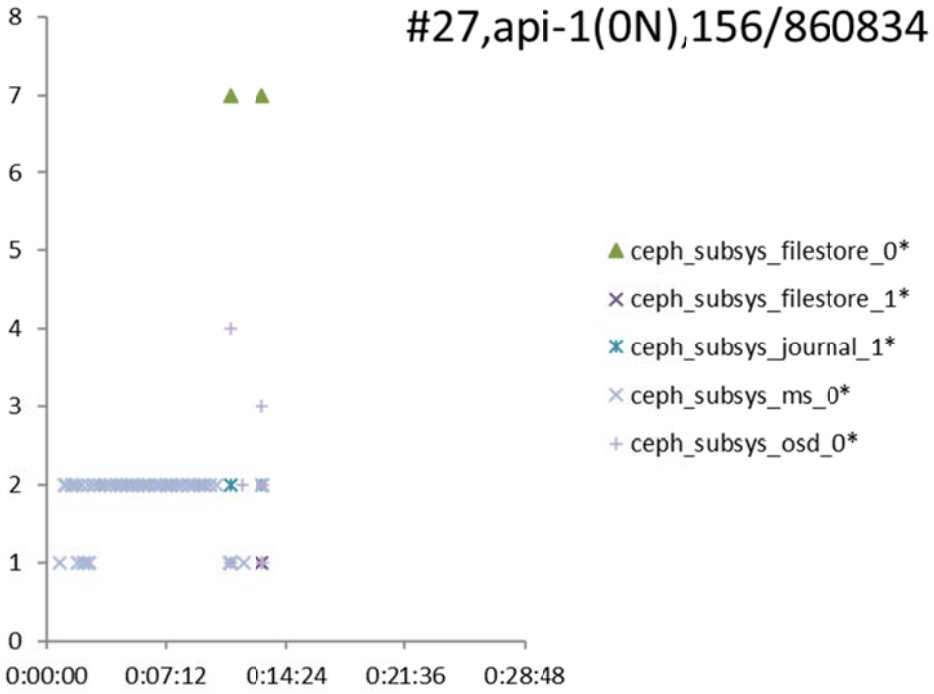
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#27 cluster NIC 故障





現象と対処：

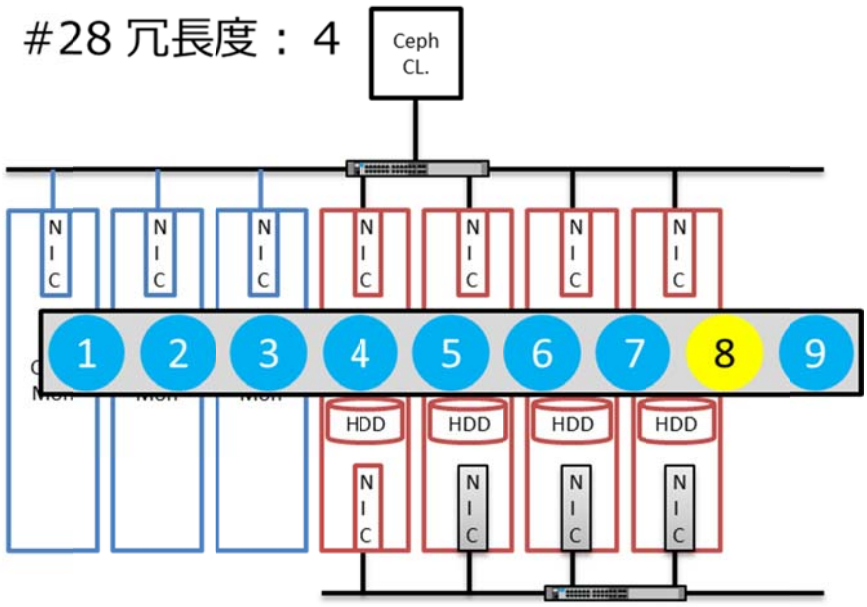
オペレーション： api-6

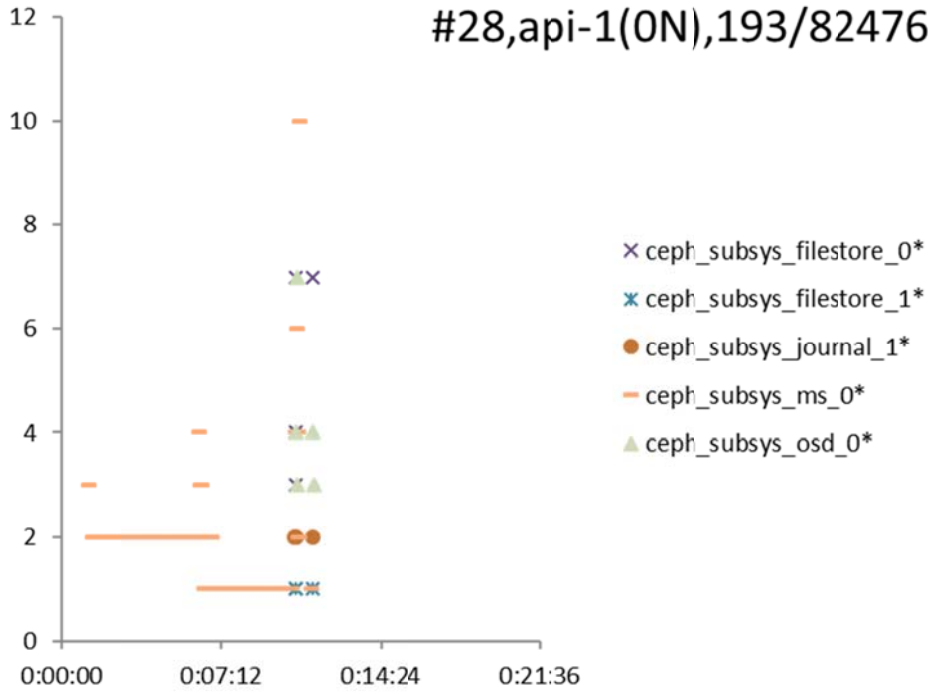
現象： スナップショット-C からインスタンス-D が起動しない (Status: BUILD, Task State: spawning)

対処：

1) インスタンス-D を削除
2) インスタンス-D を起動

#28 cluster NIC 故障





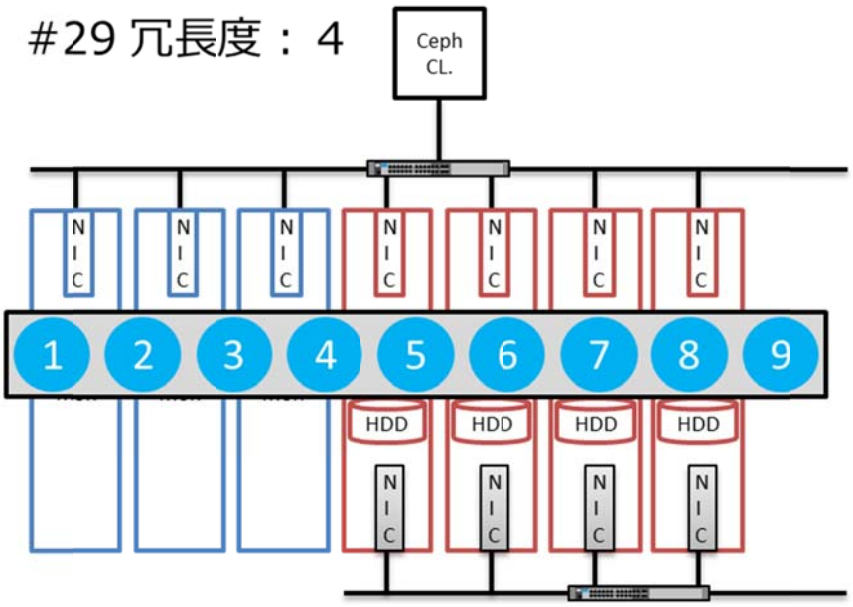
現象と対処：

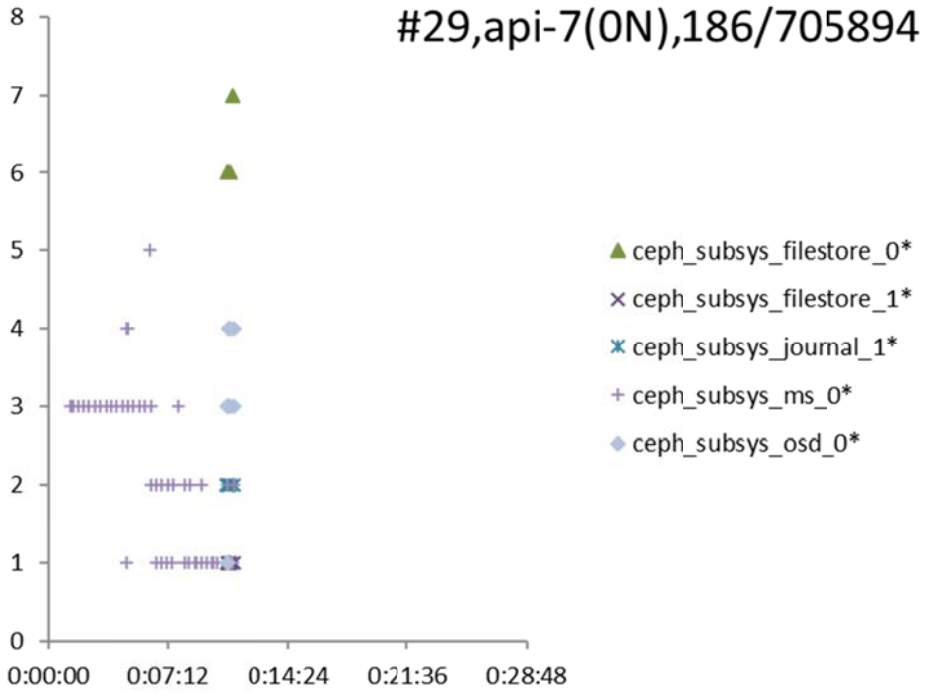
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

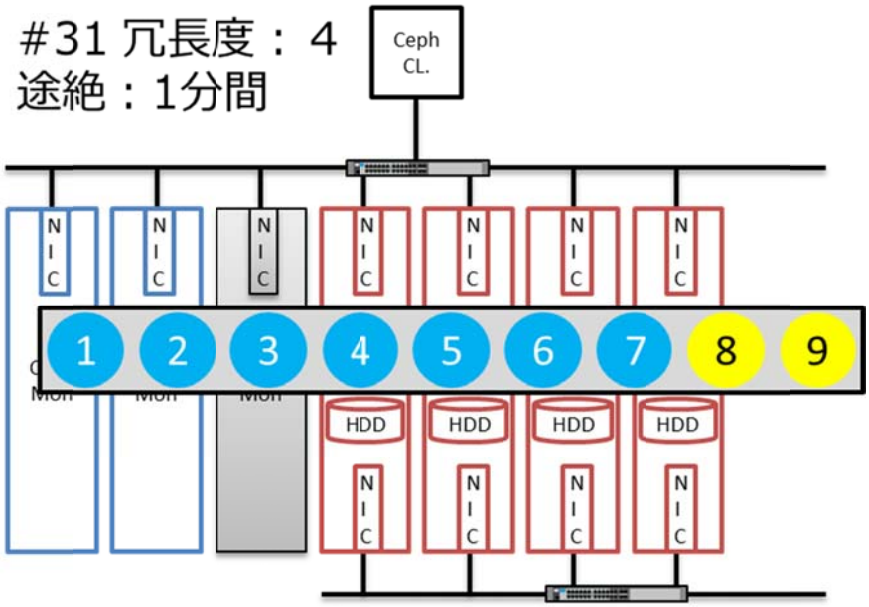
#29 cluster NIC 故障

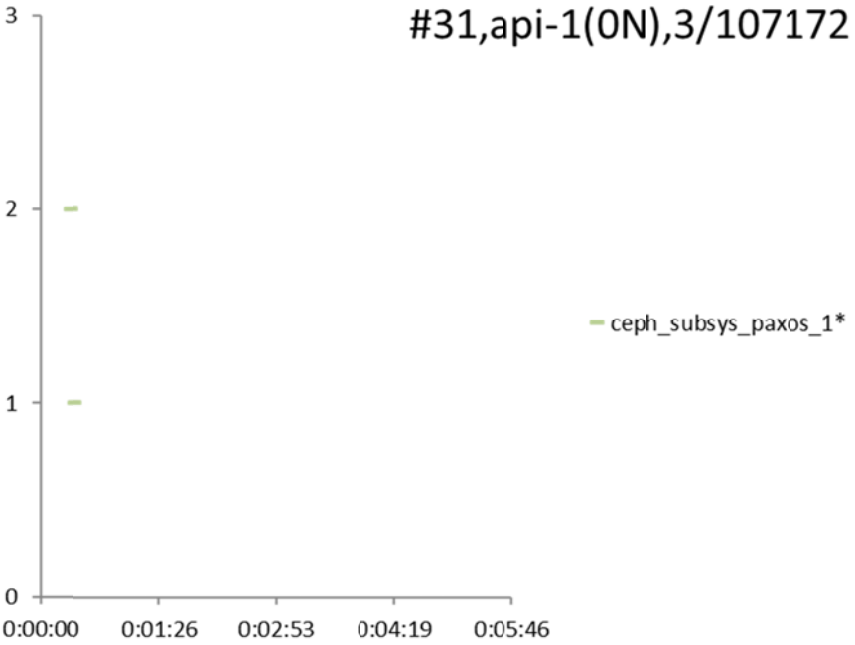




現象と対処：なし

#31 monitor ネットワーク途絶





現象と対処：

オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

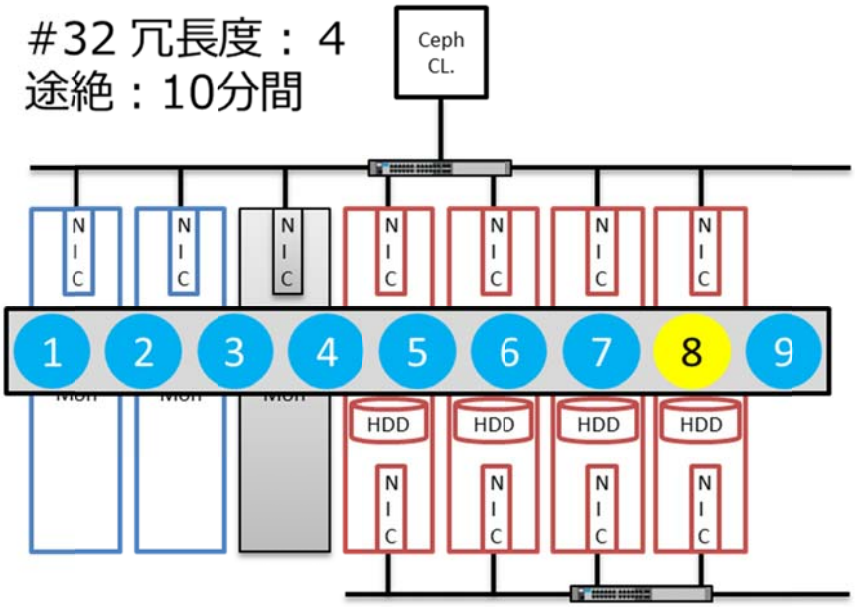
対処： スナップショット-C を削除

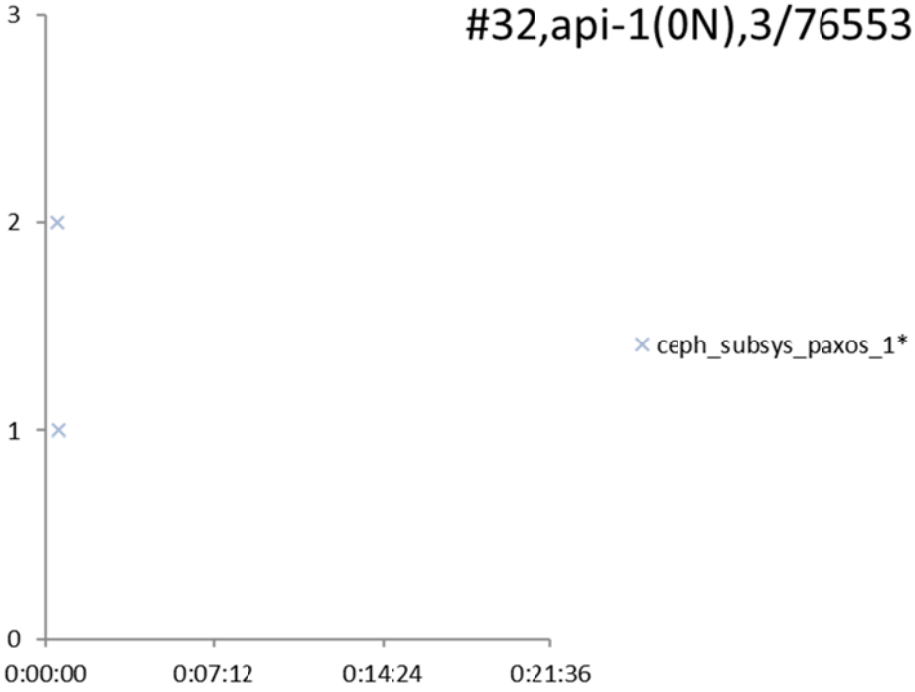
オペレーション： api-9

現象： ボリューム-B が削除されない (Status: available)

対処： ボリューム-B を削除

#32 monitor ネットワーク途絶





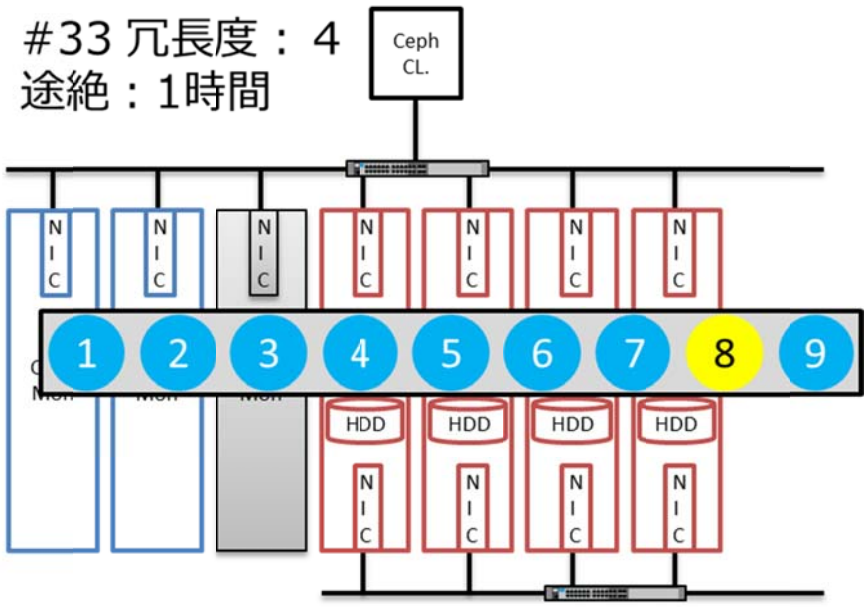
現象と対処：

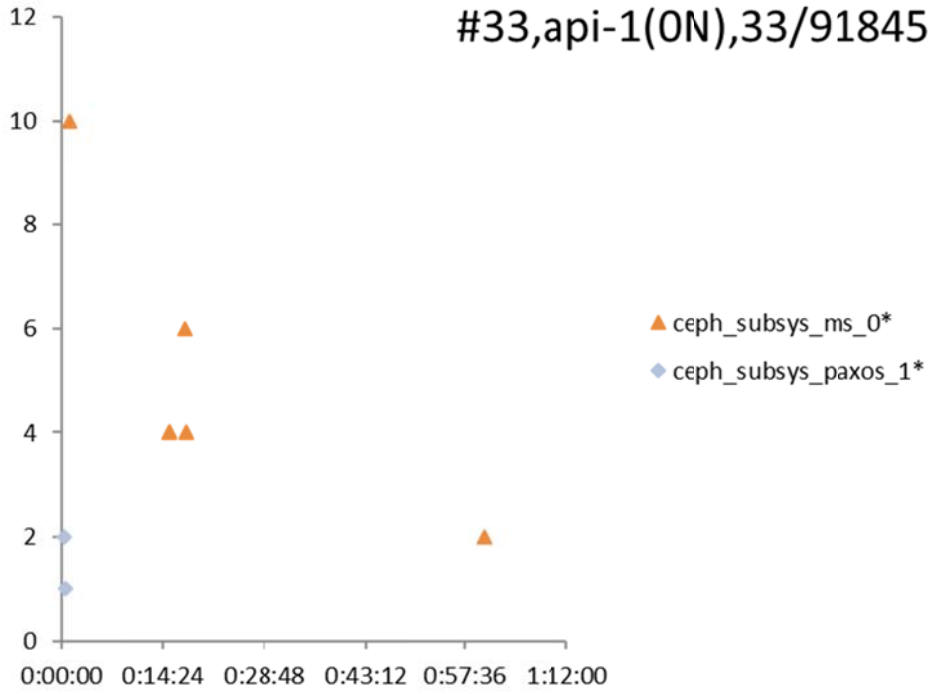
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#33 monitor ネットワーク途絶





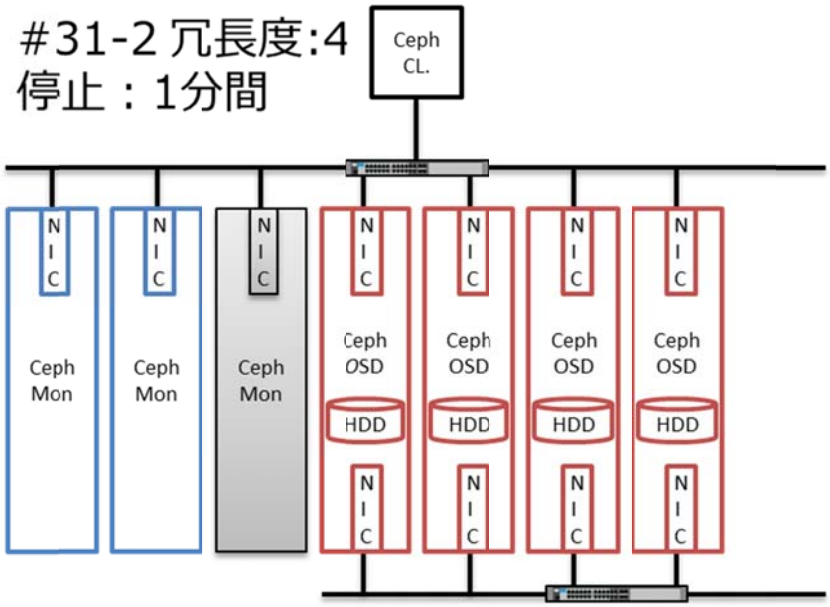
現象と対処：

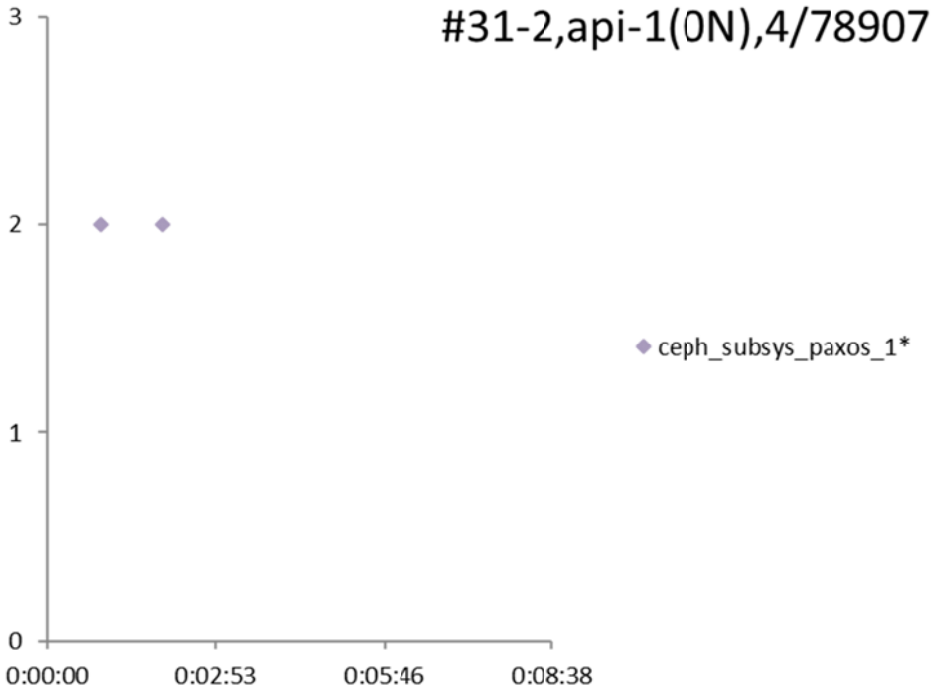
オペレーション：api-8

現象：スナップショット-C が削除されない（Status: ACTIVE)

対処：スナップショット-C を削除

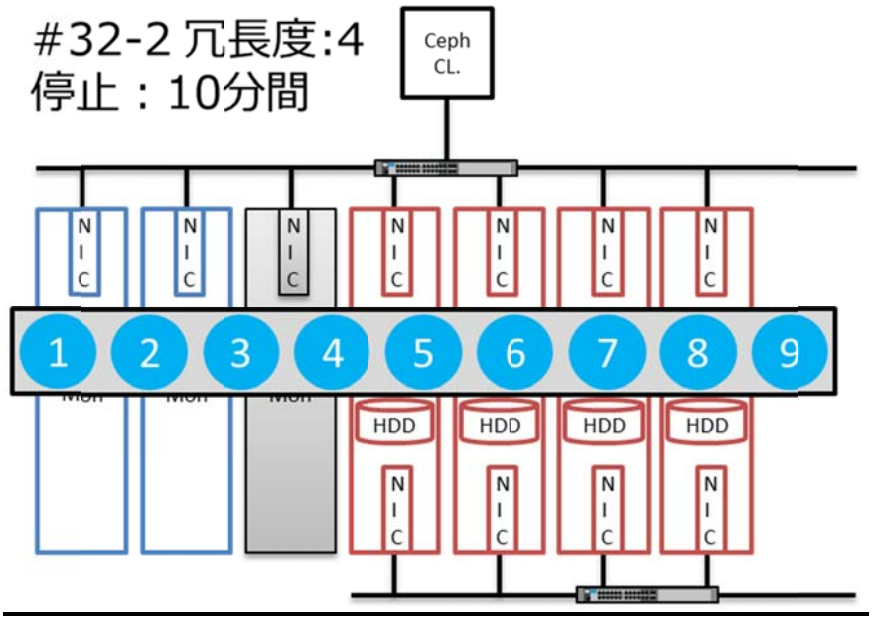
#31-2 monitor 停止

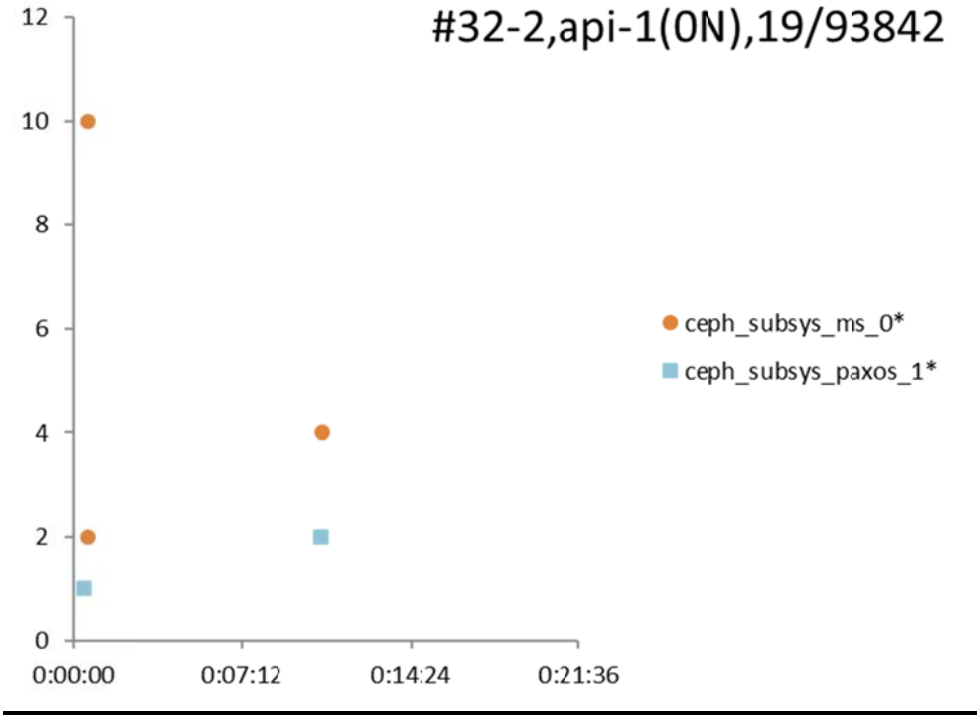




現象と対処：なし

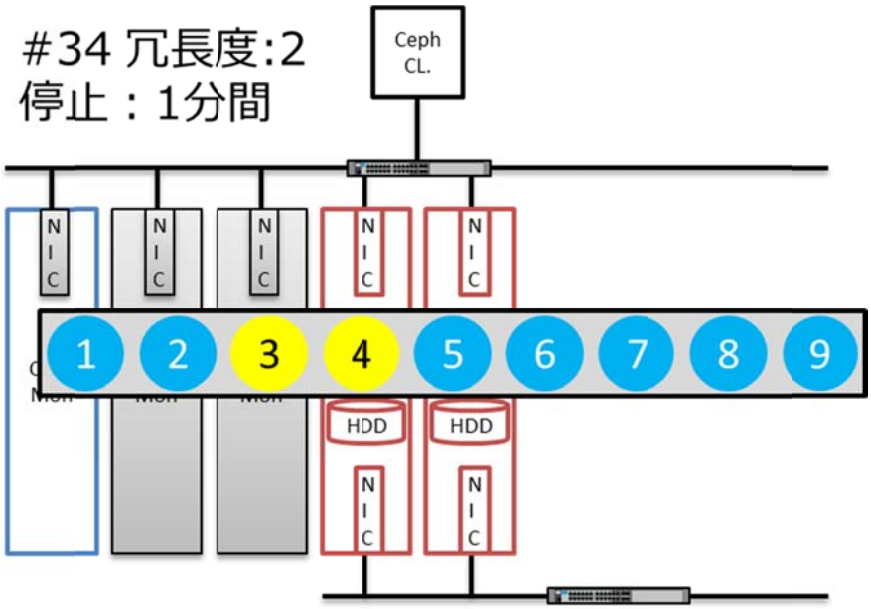
#32-2 monitor 停止

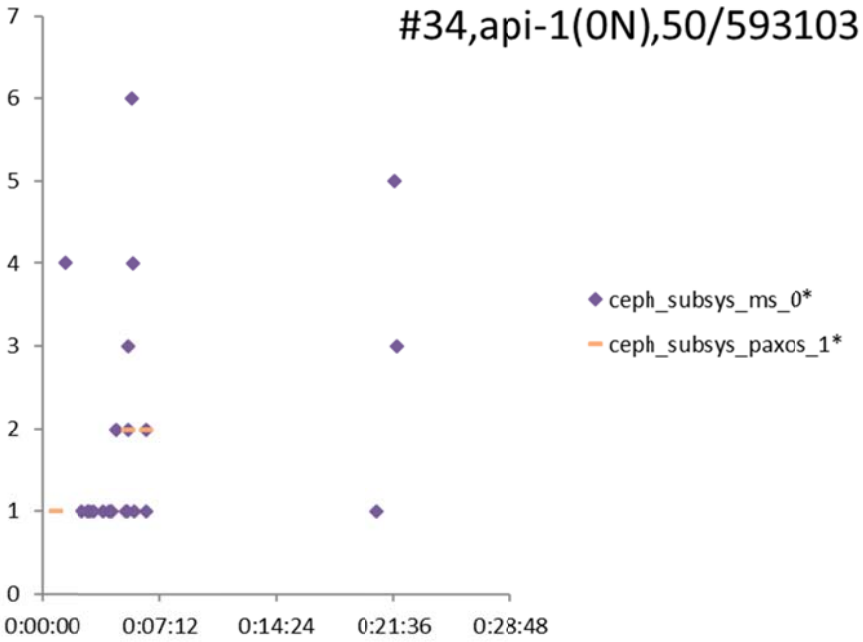




現象と対処：なし

#34 monitor 停止





現象と対処：

オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: available)

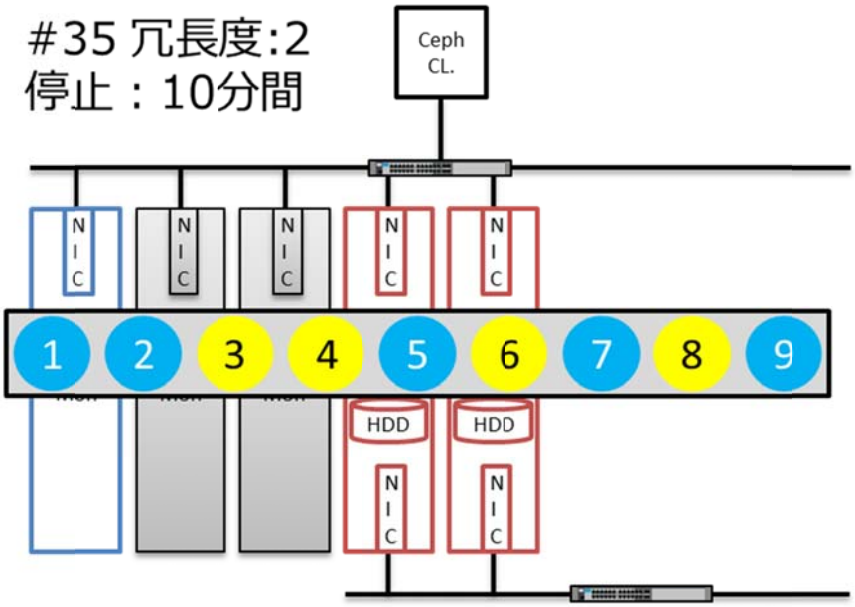
対処： ボリューム-B をインスタンス-A に接続

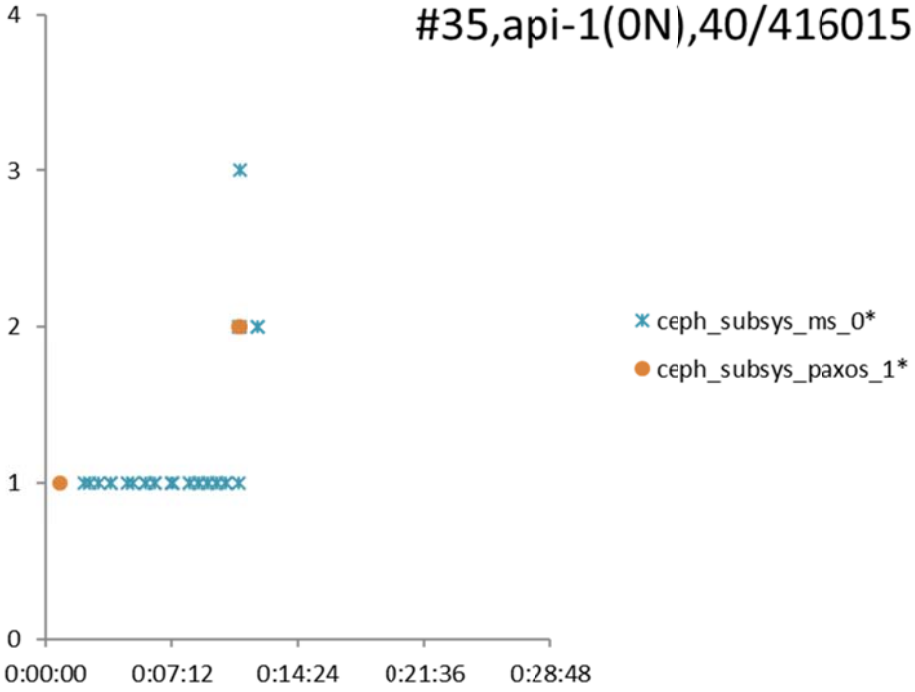
オペレーション： api-4

現象： インスタンス-A のスナップショット-C が作成されない

対処： スナップショット-C を再作成

#35 monitor 停止





現象と対処：

オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: available)

対処： ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C が作成されない

対処: スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

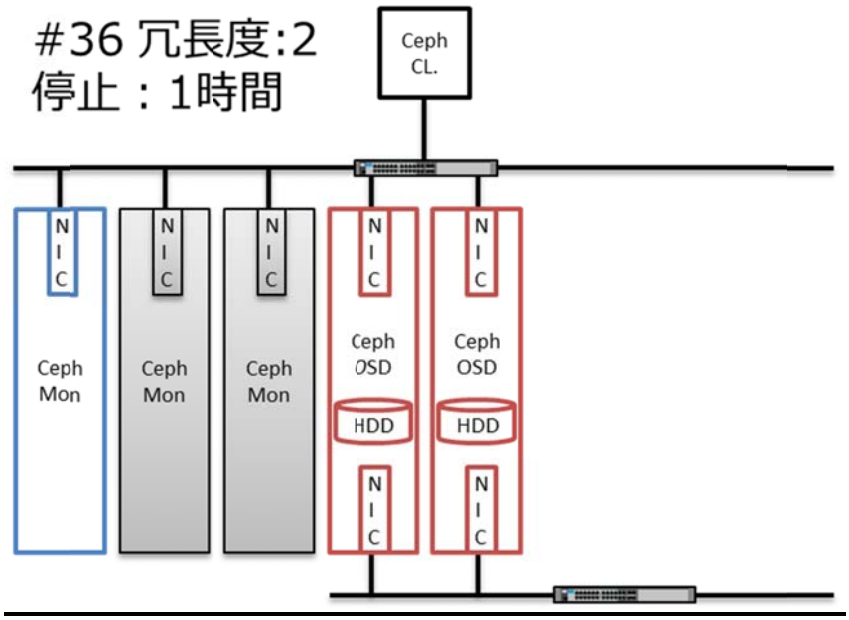
1) インスタンス-D を削除
2) インスタンス-D を起動

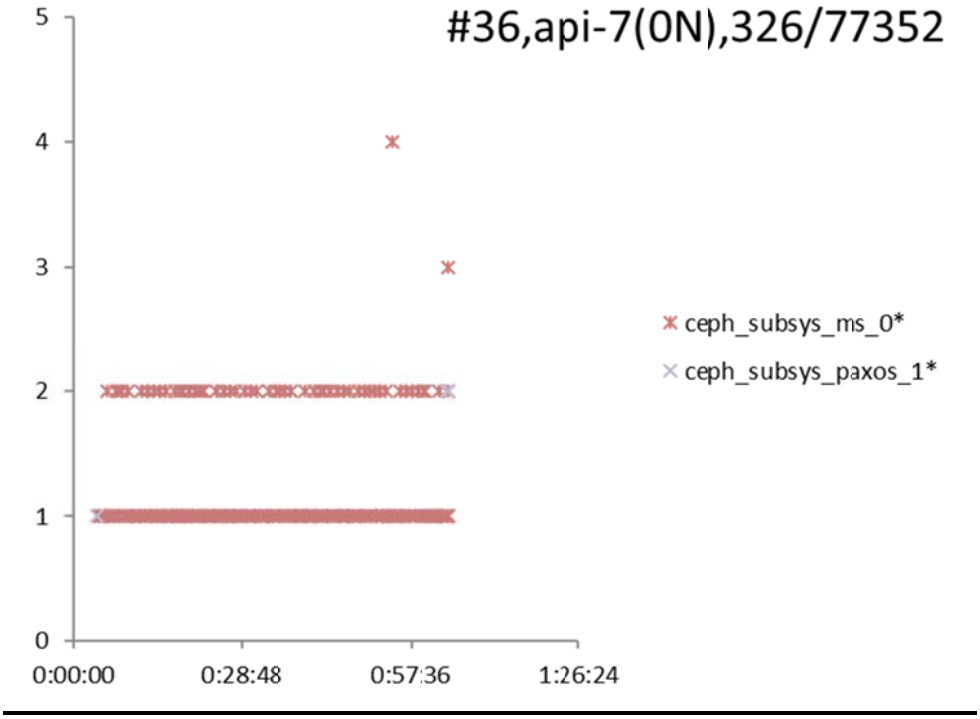
オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処: スナップショット-C を削除

#36 monitor 停止





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:ERROR, Task State: -)

対処：

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-4

現象: インスタンス-A が ACTIVE でない (Status:SHUTOFF, Task State: -) 、かつ、スナップショット-C が作成されない

対処:

1) 1) インスタンス-A を削除
2) インスタンス-A を起動
3) ボリューム-B をインスタンス-A に接続
4) スナップショット-C を再作成

オペレーション: api-5

現象: インスタンス-A が ACTIVE でない (Status:SHUTOFF, Task State: -)

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動
3) ボリューム-B をインスタンス-A に接続
4) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処: "glance" データベースの "images" テーブル上の "name" の該当するエントリの "deleted" の値を 0 から 1 に変更する。

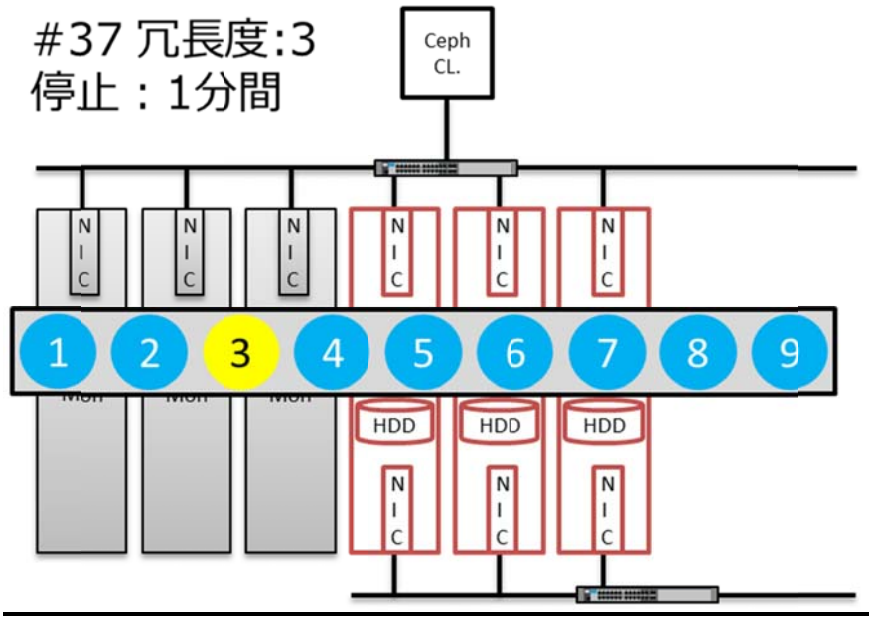
オペレーション: api-9

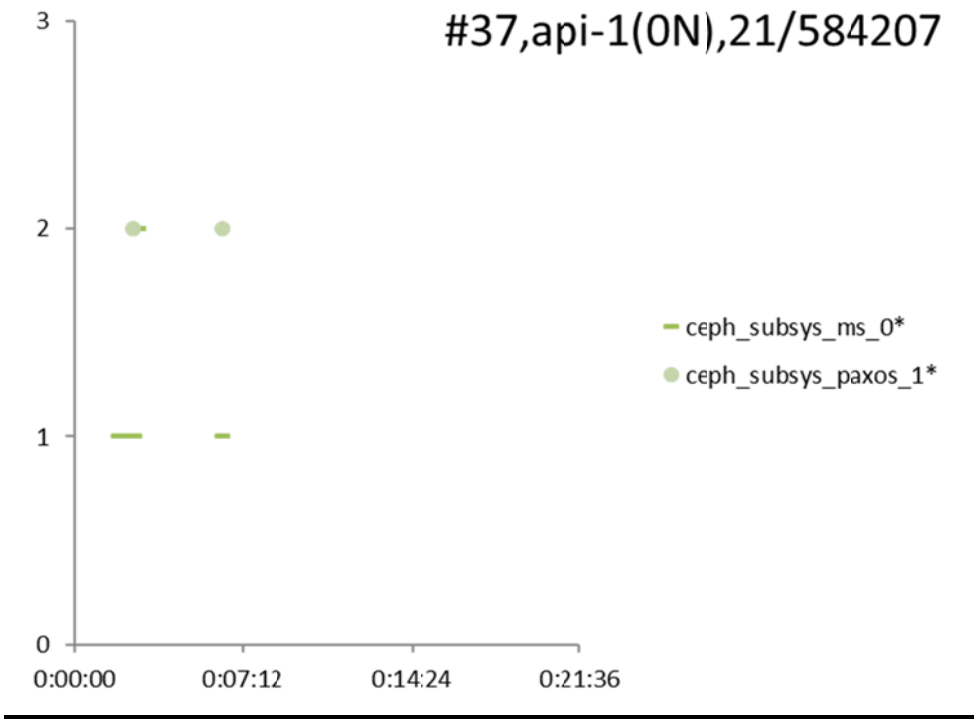
現象: ボリューム-B が削除されない (Status: error_deleting)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B を削除

#37 monitor 停止





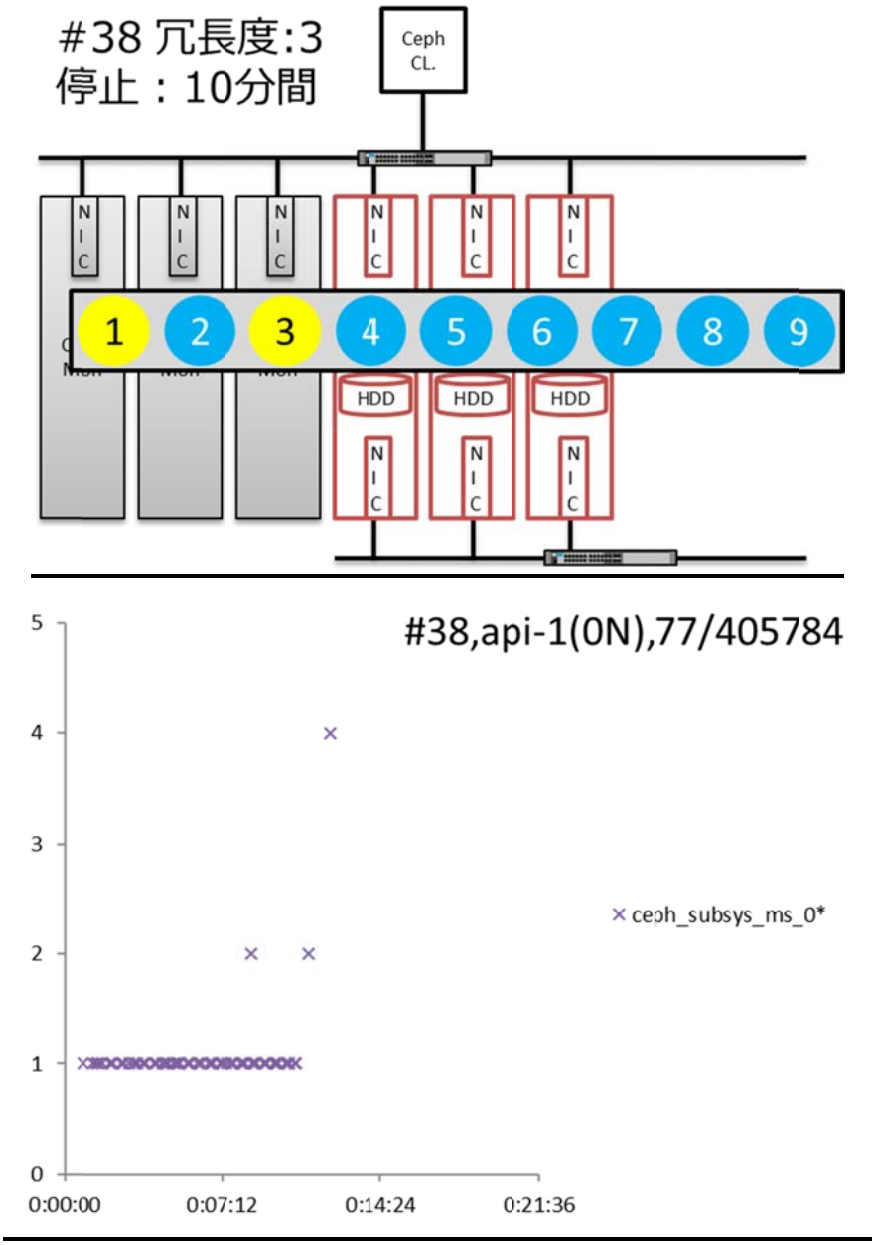
現象と対処：

オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: available)

対処： ボリューム-B をインスタンス-A に接続

#38 monitor 停止



現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:ERROR, Task State: -)

対処：

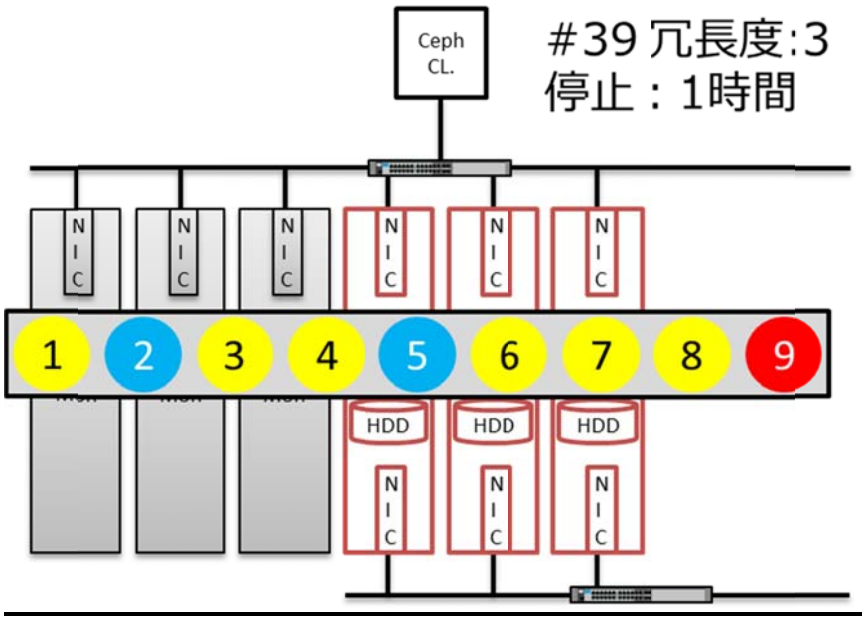
1) インスタンス-A を削除
2) インスタンス-A を起動

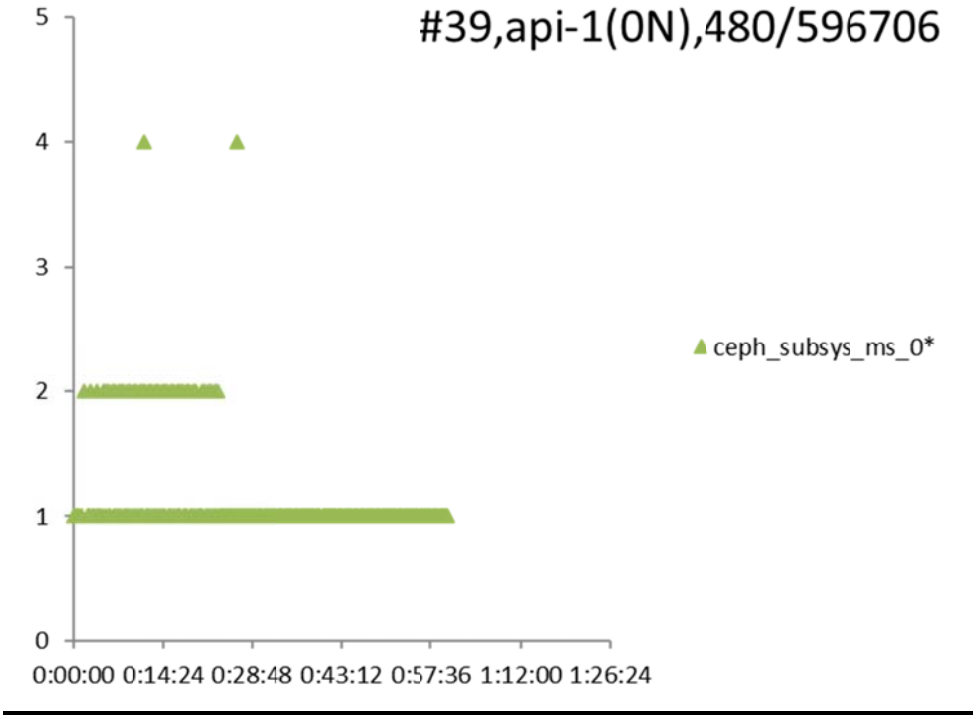
オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: available)

対処： ボリューム-B をインスタンス-A に接続

#39 monitor 停止





現象と対処:

オペレーション: api-1

現象: インスタンス-A が起動しない (Status:ERROR, Task State: -)

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: available)

対処: ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A が ACTIVE でない (Status:SHUTOFF, Task State: -) 、かつ、スナップショット-C が作成されない

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動
3) ボリューム-B をインスタンス-A に接続
4) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処: スナップショット-C を削除

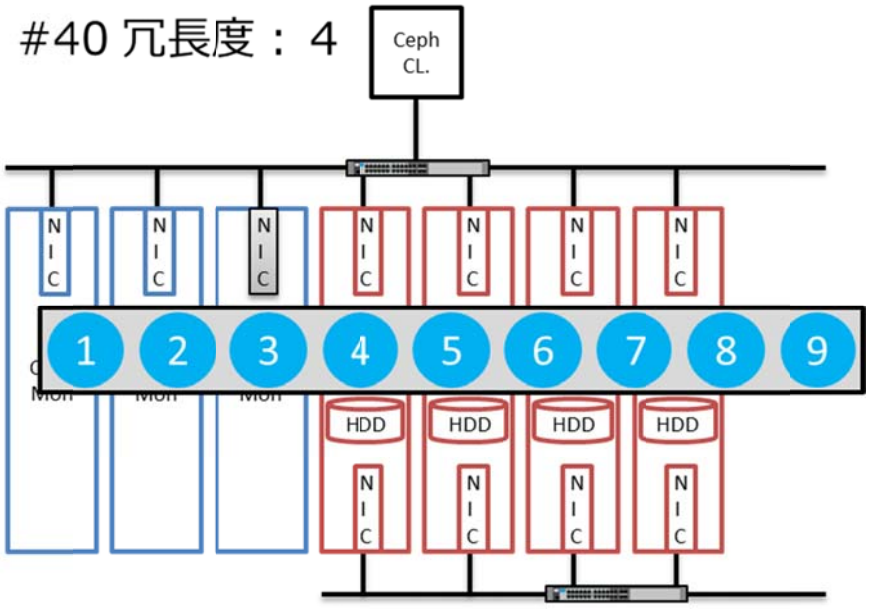
オペレーション: api-9

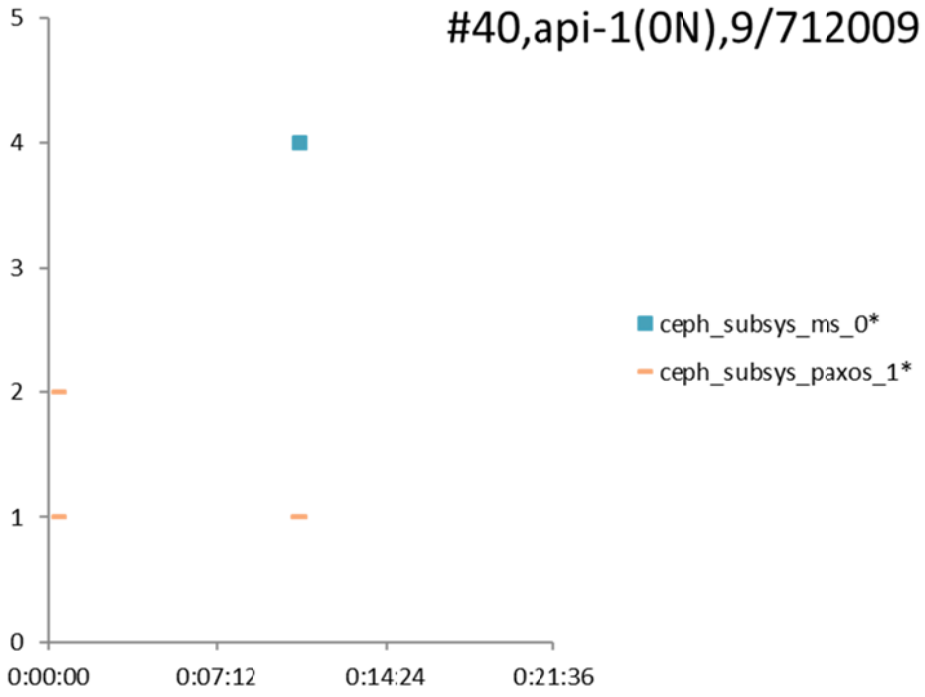
現象: ボリューム-B が削除されない (Status: error_deleting)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B を削除

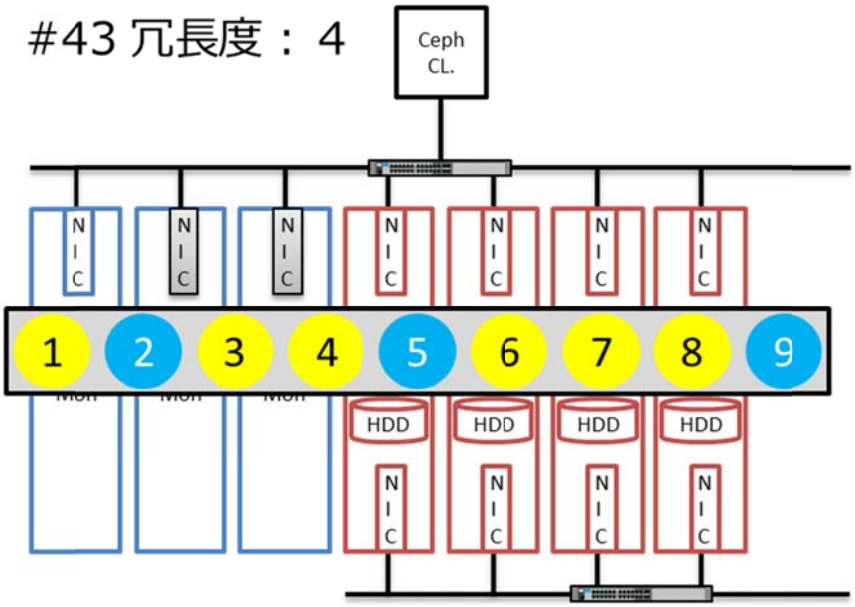
#40 public NIC 故障 (2)

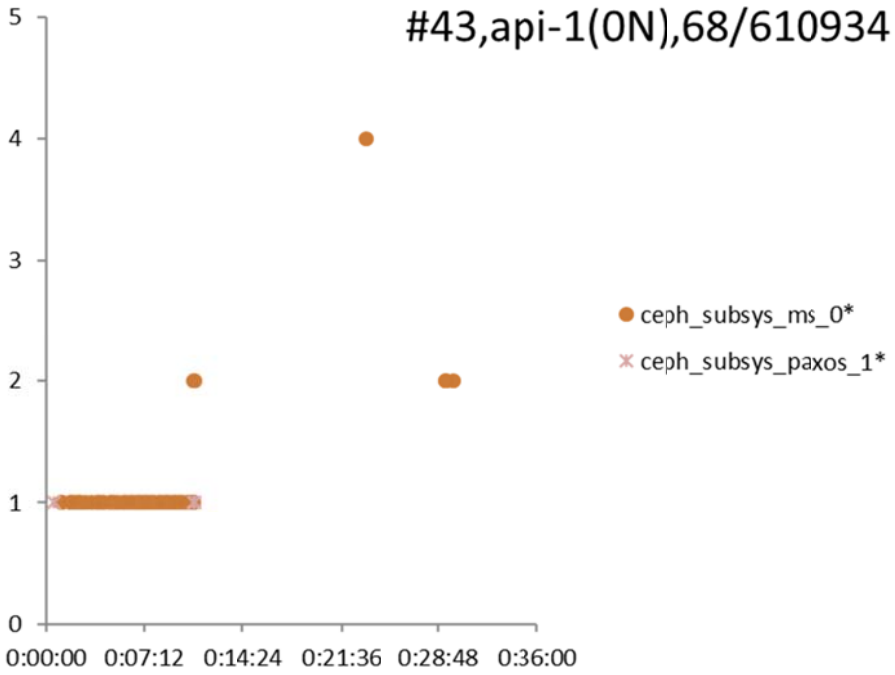




現象と対処：なし

#43 public NIC 故障 (2)





現象と対処：

オペレーション： api-1

現象： インスタンス-A が起動しない (Status:ERROR, Task State: -)

対処：

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: available)

対処: ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C が作成されない

対処: スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

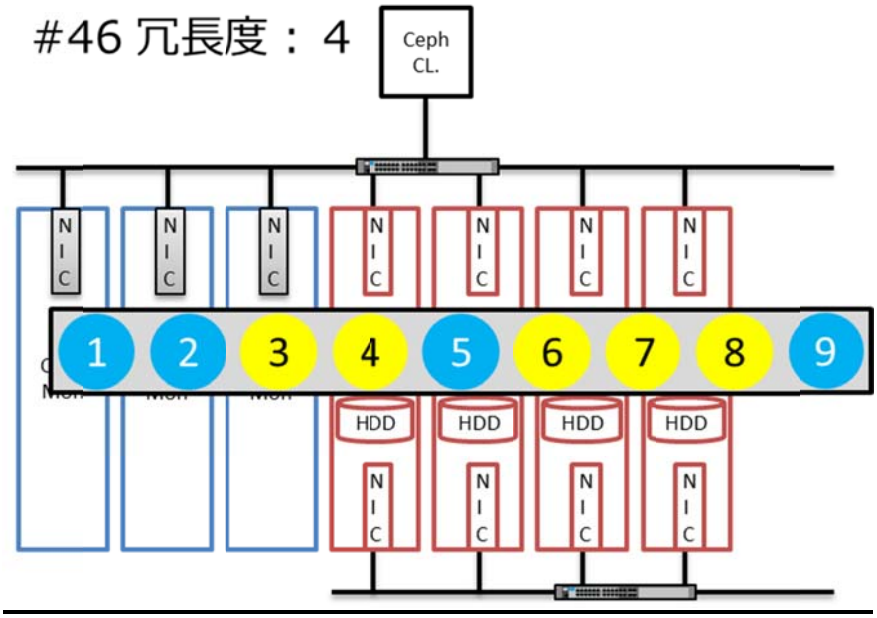
対処: インスタンス-D を削除

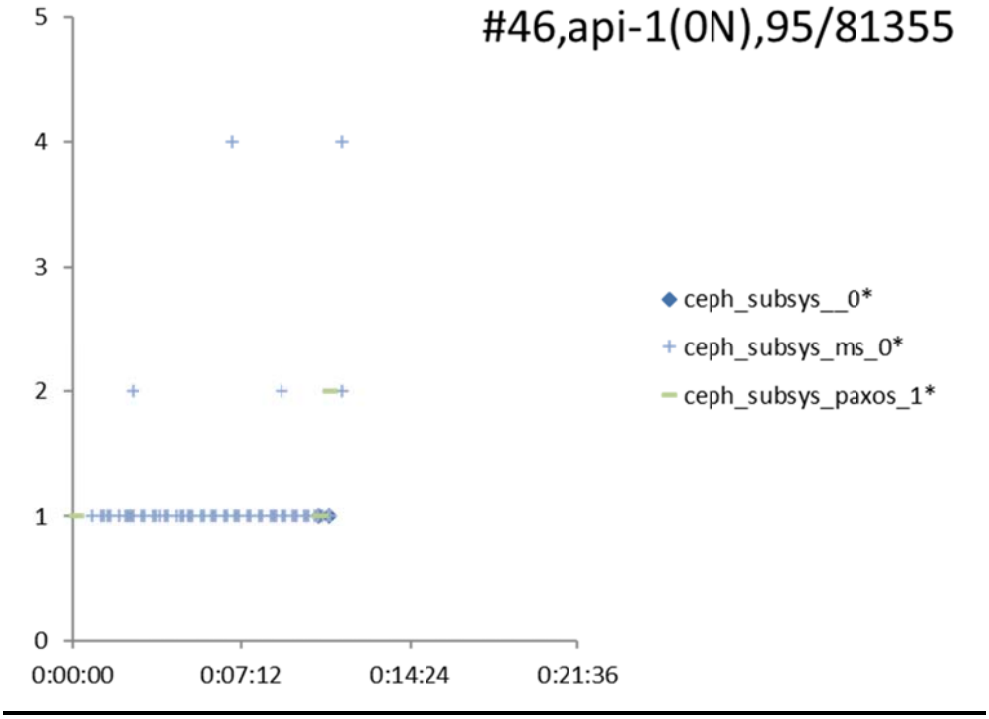
オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処 :スナップショット-C を削除

#46 public NIC 故障 (2)





現象と対処：

オペレーション： api-3

現象： ボリューム-B がインスタンス-A に接続しない (Status: available)

対処： ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A のスナップショット-C が作成されない

対処: スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

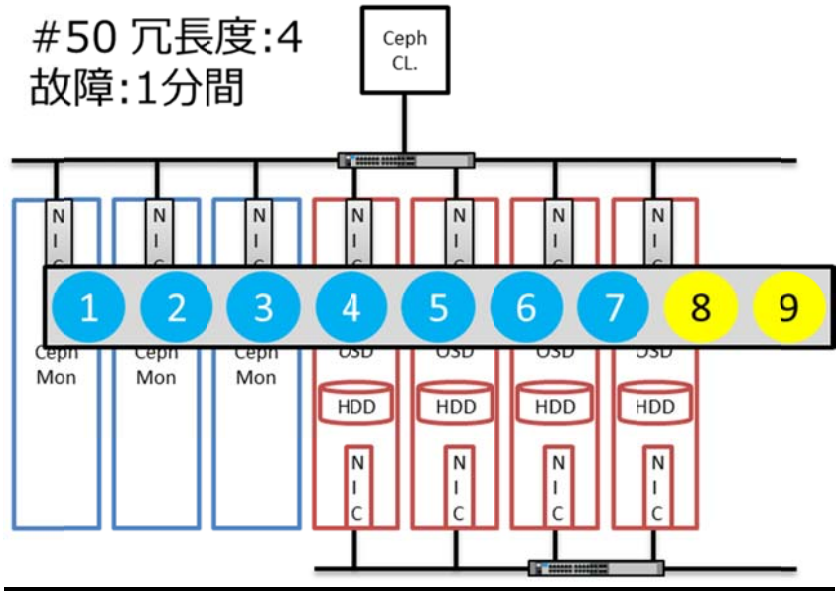
対処: インスタンス-D を削除

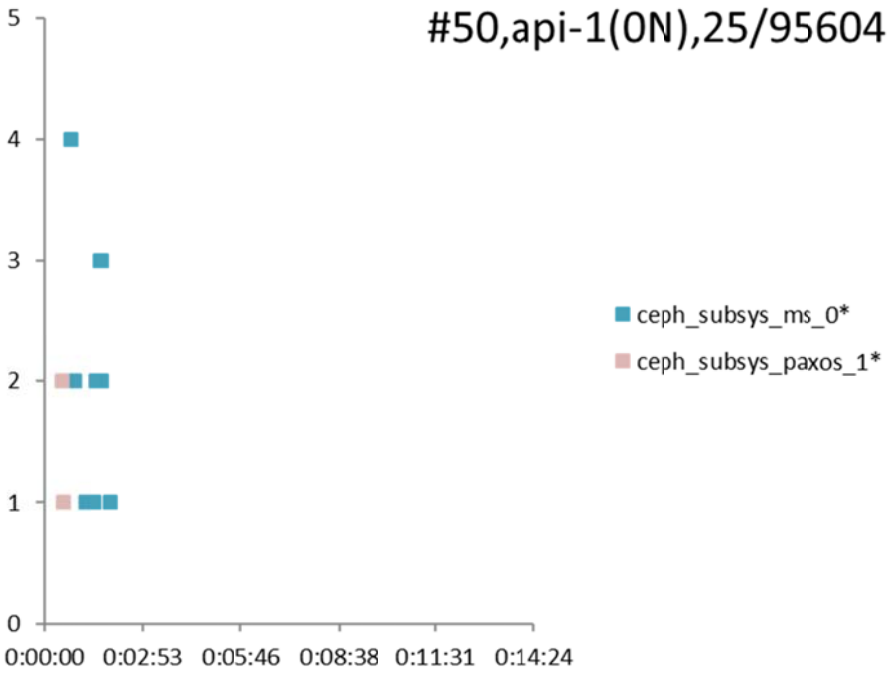
オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処: スナップショット-C を削除

#50 スイッチ故障





現象と対処:

オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

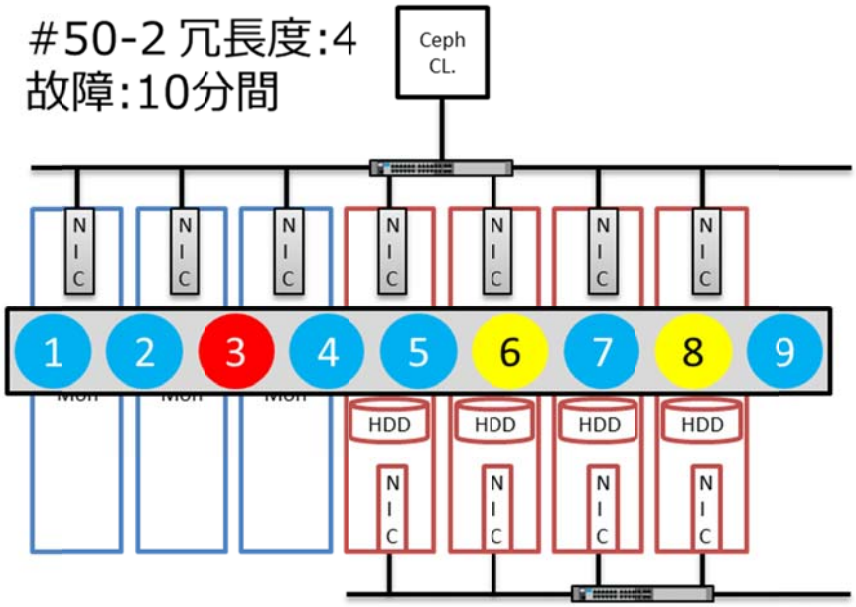
対処: スナップショット-C を削除

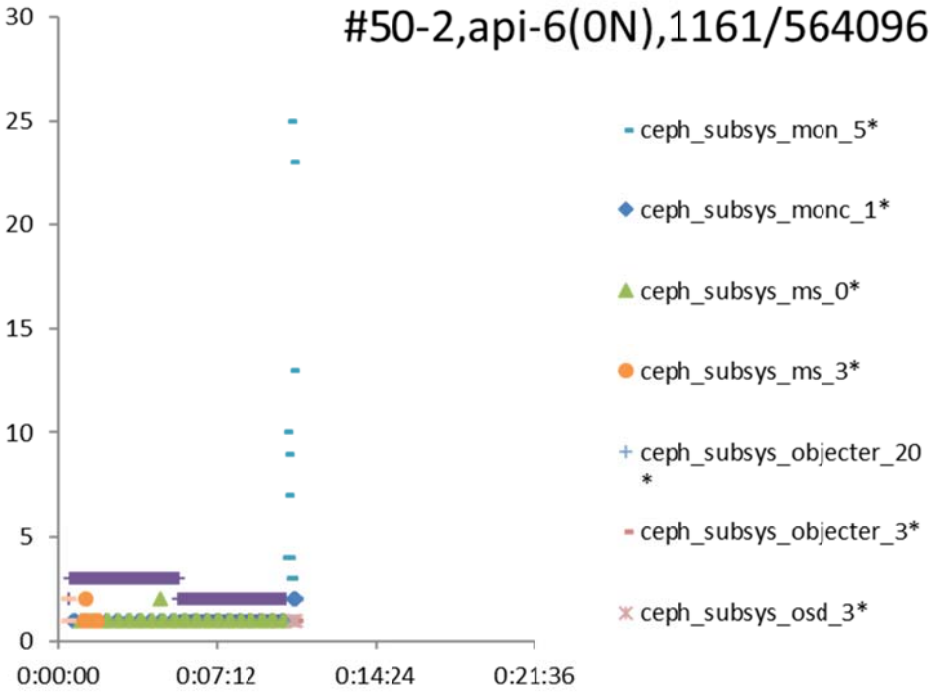
オペレーション: api-9

現象: ボリューム-B が削除されない (Status: available)

対処: ボリューム-B を削除

#50-2 スイッチ故障





現象と対処:

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: available)

対処: ボリューム-B をインスタンス-A に接続 (/dev/vdb 引数を省略する必要がある)

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

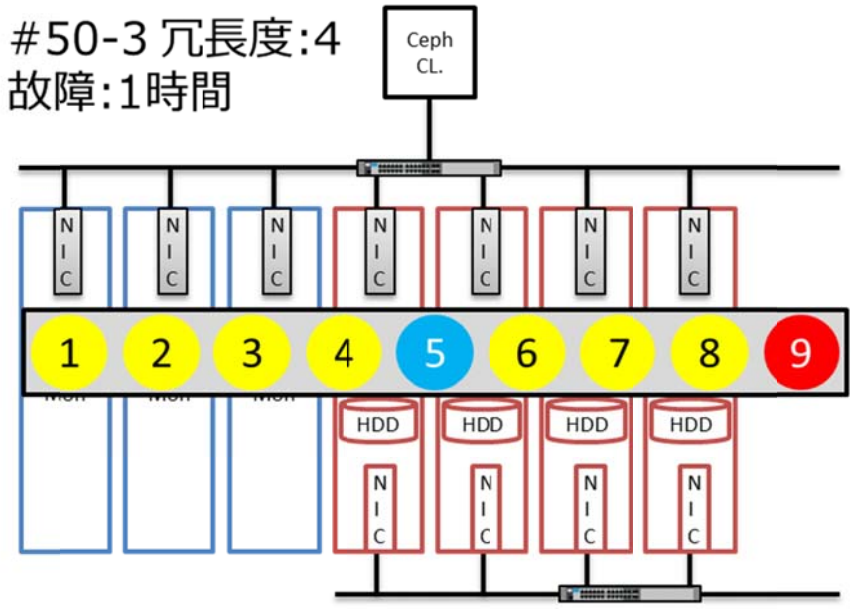
1) インスタンス-D を削除
2) インスタンス-D を起動

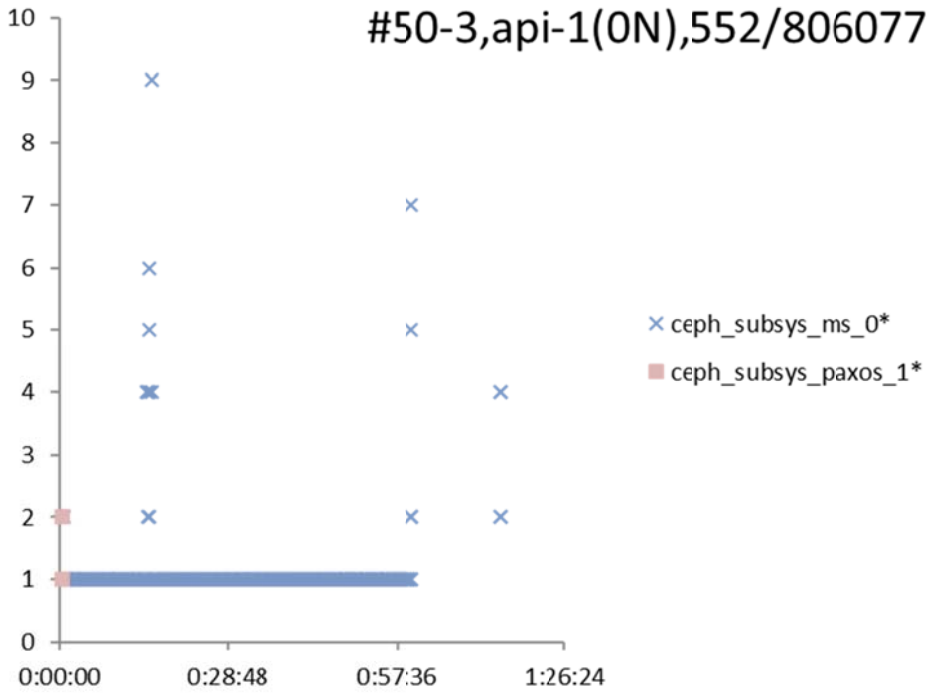
オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処: スナップショット-C を削除

#50-3 スイッチ故障





現象と対処:

オペレーション: api-1

現象: インスタンス-A が起動しない (Status:ERROR, Task State: -)

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動

オペレーション: api-2

現象: ボリューム-B が作成されない (Status: creating)

対処:

1) ボリューム-B を削除
2) ボリューム-B を再作成

オペレーション: api-3

現象: ボリューム-B がインスタンス-A に接続しない (Status: available)

対処: ボリューム-B をインスタンス-A に接続

オペレーション: api-4

現象: インスタンス-A が ACTIVE でない (Status:SHUTOFF, Task State: -) 、かつ、スナップショット-C が作成されない

対処:

1) インスタンス-A を削除
2) インスタンス-A を起動
3) ボリューム-B をインスタンス-A に接続
4) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C が削除されない (Status: ACTIVE)

対処: スナップショット-C を削除

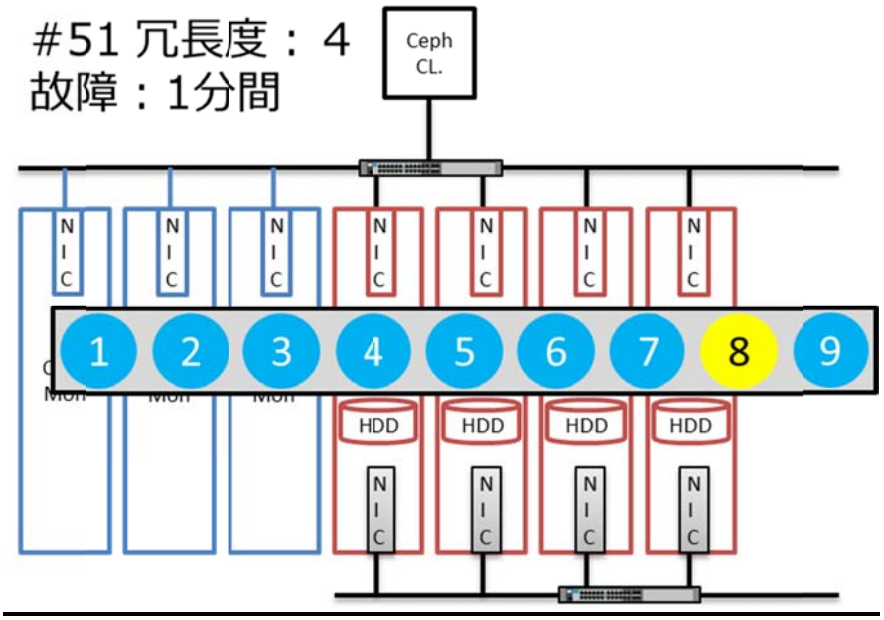
オペレーション: api-9

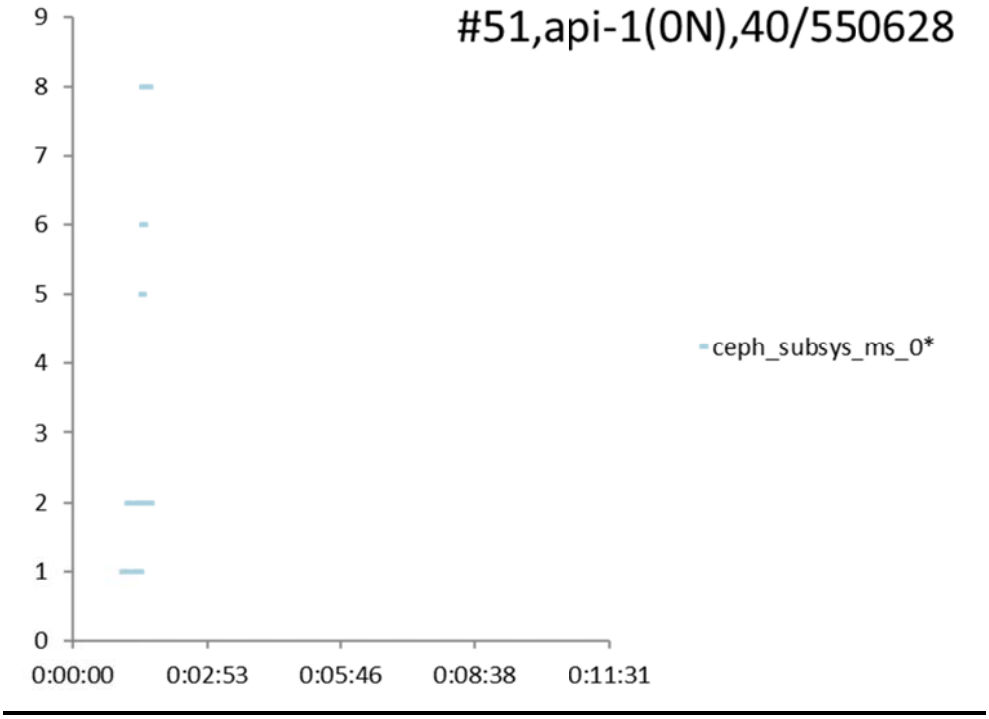
現象: ボリューム-B が削除されない (Status: error_deleting)

対処:

1) "cinder" データベースの "volumes" テーブル上の "id" の該当するエントリの "status" の値を "available" に変更する。
2) ボリューム-B を削除

#51 スイッチ故障





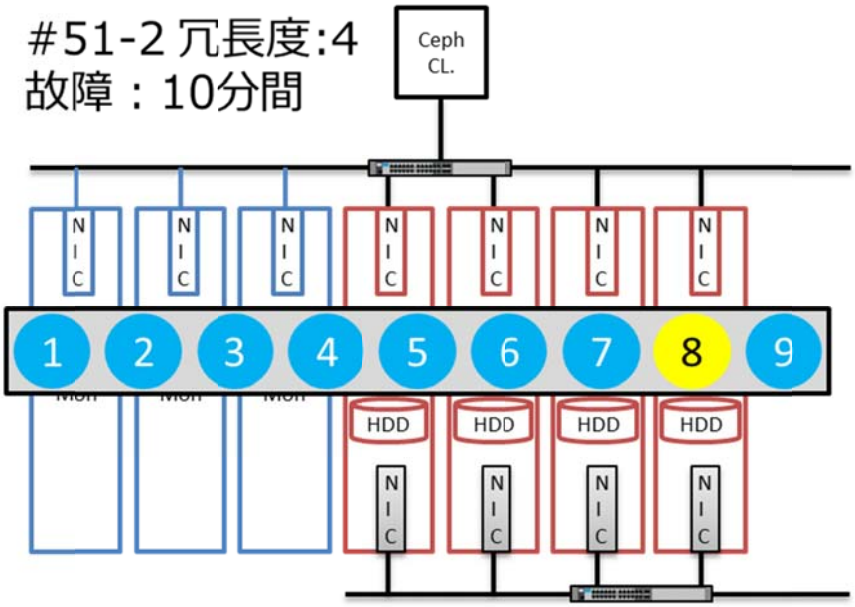
現象と対処：

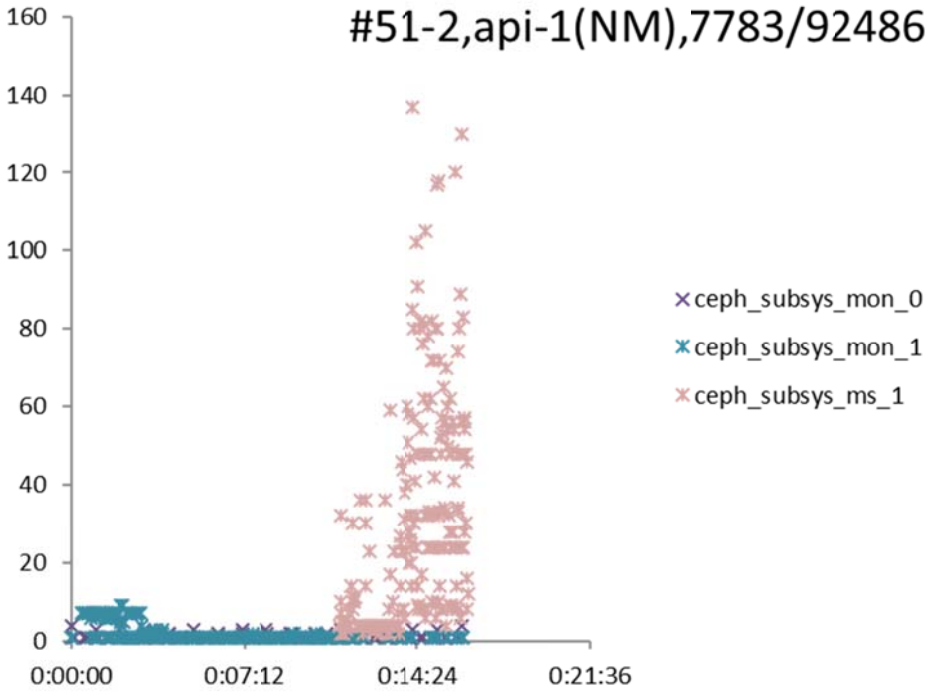
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#51-2 スイッチ故障





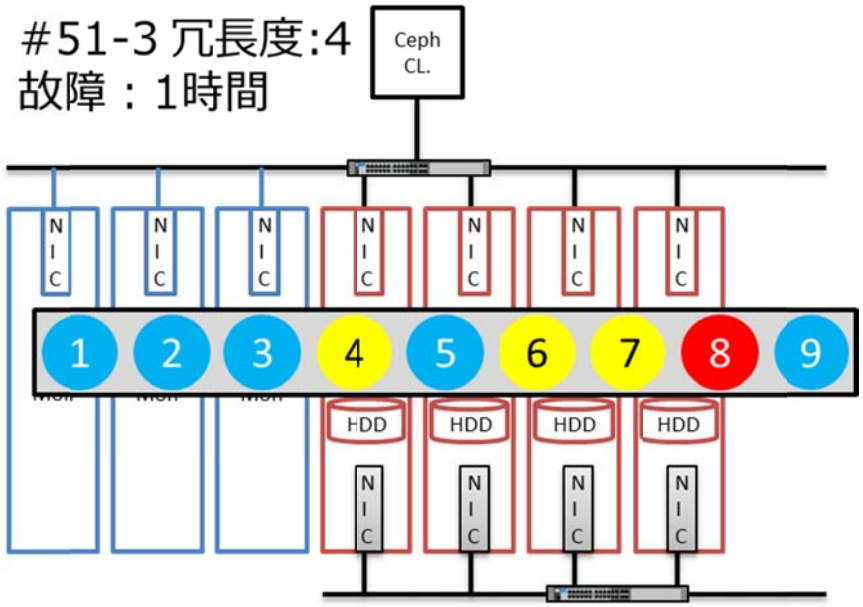
現象と対処：

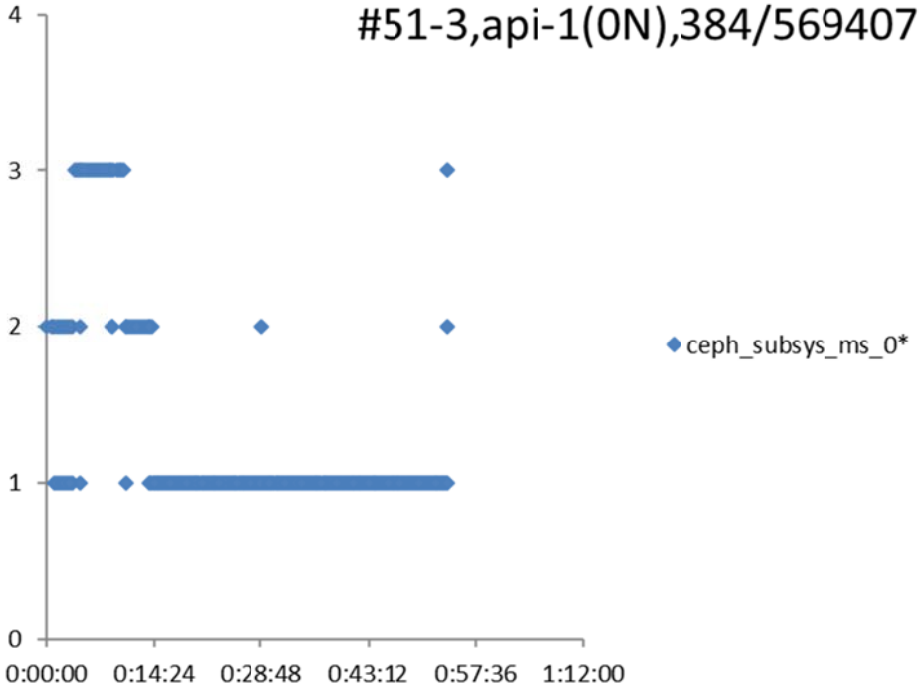
オペレーション： api-8

現象： スナップショット-C が削除されない (Status: ACTIVE)

対処： スナップショット-C を削除

#51-3 スイッチ故障





現象と対処：

オペレーション： api-4

現象： インスタンス-A のスナップショット-C の作成が完了しない (Status: SAVING、Task State: -)

対処：

1) スナップショット-C を削除
2) スナップショット-C を再作成

オペレーション: api-6

現象: スナップショット-C からインスタンス-D が起動しない (Status: ERROR, Task State: -)

対処:

1) インスタンス-D を削除
2) インスタンス-D を起動

オペレーション: api-7

現象: インスタンス-D が削除されない (Status: ERROR, Task State: -)

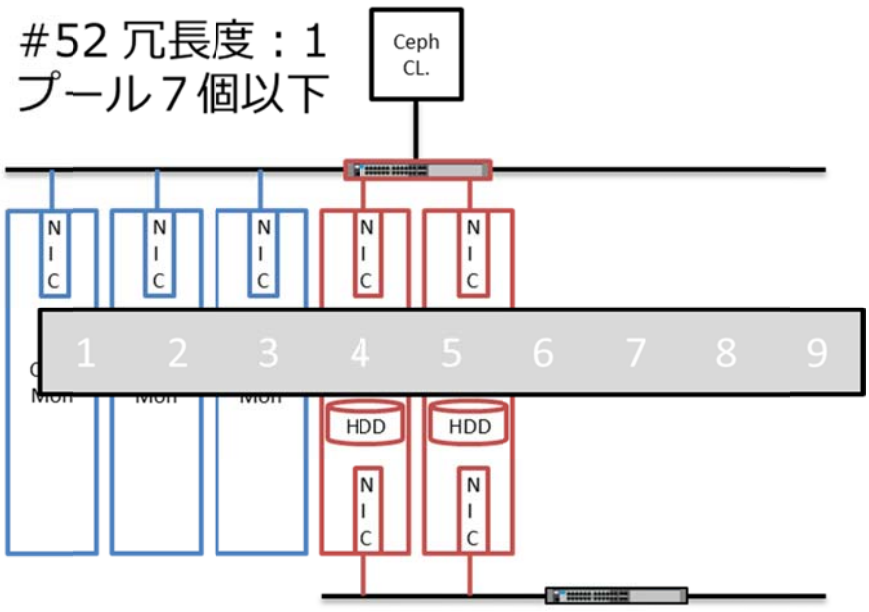
対処: インスタンス-D を削除

オペレーション: api-8

現象: スナップショット-C の削除が完了しない (Status: DELETED)

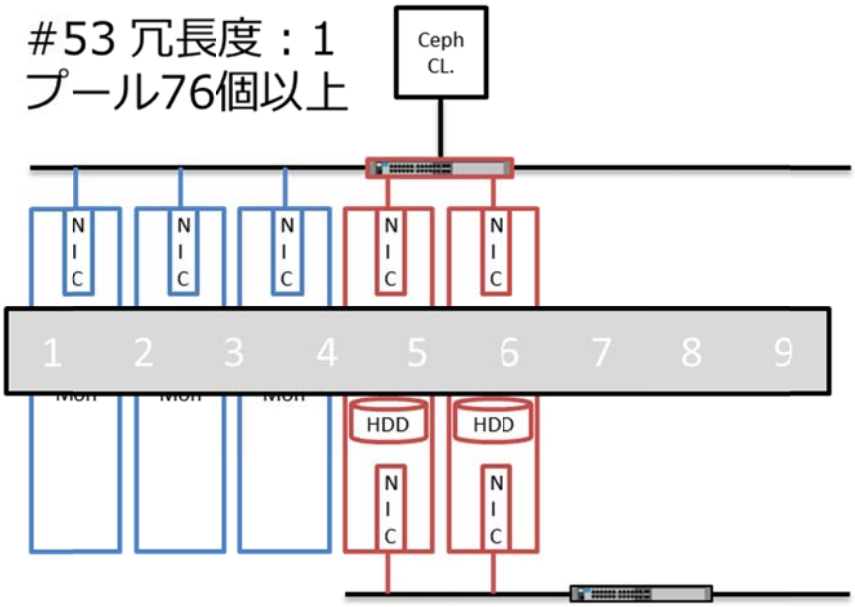
対処: "glance" データベースの "images" テーブル上の "name" の該当するエントリの "deleted" の値を 0 から 1 に変更する。

#52 too few PGs per OSD



Ceph OSD 台数あたりの PG 数が少なすぎる場合については組み合わせから除外しました。OpenStack 環境に対するオペレーションへの影響がないことが明らかなことが除外した主な理由です。

#53 too many PGs per OSD



Ceph OSD 台数あたりの PG 数が多すぎる場合については組み合わせから除外しました。OpenStack 環境に対するオペレーションへの影響がないことが明らかなことが除外した主な理由です。

OpenStack/Ceph 異常系テスト (2) の結果の最後に、特徴ログの出力に必要なログレベル設定 (サブシステムおよびログレベルの組み合わせ) について述べます。

「4.3. ログの分析」で述べたように、特徴ログの出力に必要なログレベル設定 (サブシステムおよびログレベルの組み合わせ) を求めるため、障害パターン毎に選択した特徴ログ (ON ログまたはログレベル 1 以下の NM ログ) に基づいて、当該ログを出力しているサブシステムおよびログレベルの組み合わせを抽出し、その総和を求めます。

	-1	0	1	2	3	4	5	6	7	10	11	12	14	15	20	21	25	30	35	40
ceph_subsys_		**																		
ceph_subsys_asok							*													
ceph_subsys_auth										*					*			*		
ceph_subsys_civetweb																				
ceph_subsys_client																				
ceph_subsys_compressor																				
ceph_subsys_crush																				
ceph_subsys_crypto																				
ceph_subsys_filer																				
ceph_subsys_filestore		**	**				**			*				*	*					
ceph_subsys_finisher										*										
ceph_subsys_heartbeatmap			**				9/12													
ceph_subsys_javaclient																				
ceph_subsys_journal			**				**			*				**	*					
ceph_subsys_journaler																				
ceph_subsys_keyvaluestore										**				*						
ceph_subsys_mds										*										
ceph_subsys_mon		**	**	*		**	**		*	*	*			*	*		*	*		
ceph_subsys_monc		**	**							*										
ceph_subsys_ms		**	**	*	**		9/9													
ceph_subsys_newstore							**			*										
ceph_subsys_objectcacher																				
ceph_subsys_objecter					**										**					
ceph_subsys_optracker							9/9 *													
ceph_subsys_osd		**	*		**		*		*	*				*	*		*	*		
ceph_subsys_paxos			**				*		*	*					*					
ceph_subsys_rados																				
ceph_subsys_rbd										*					*					
ceph_subsys_refs		9/9																		
ceph_subsys_rgw																				
ceph_subsys_striper																				
ceph_subsys_throttle			9/1							**										
ceph_subsys_timer										*					*					
ceph_subsys_tp										*		**		*	*					
ceph_subsys_xio																				

*印箇所は、このログレベルのログがNMログとして抽出されたことを表す。

**印箇所は、このログレベルのログを特徴ログ（ONログまたはログレベル１以下のNMログ）として採用したことを表す。

日付箇所は、メモリログがサイクルするまでの期間を延長するために、メモリログレベルを最大（40）から下げたことを表す。

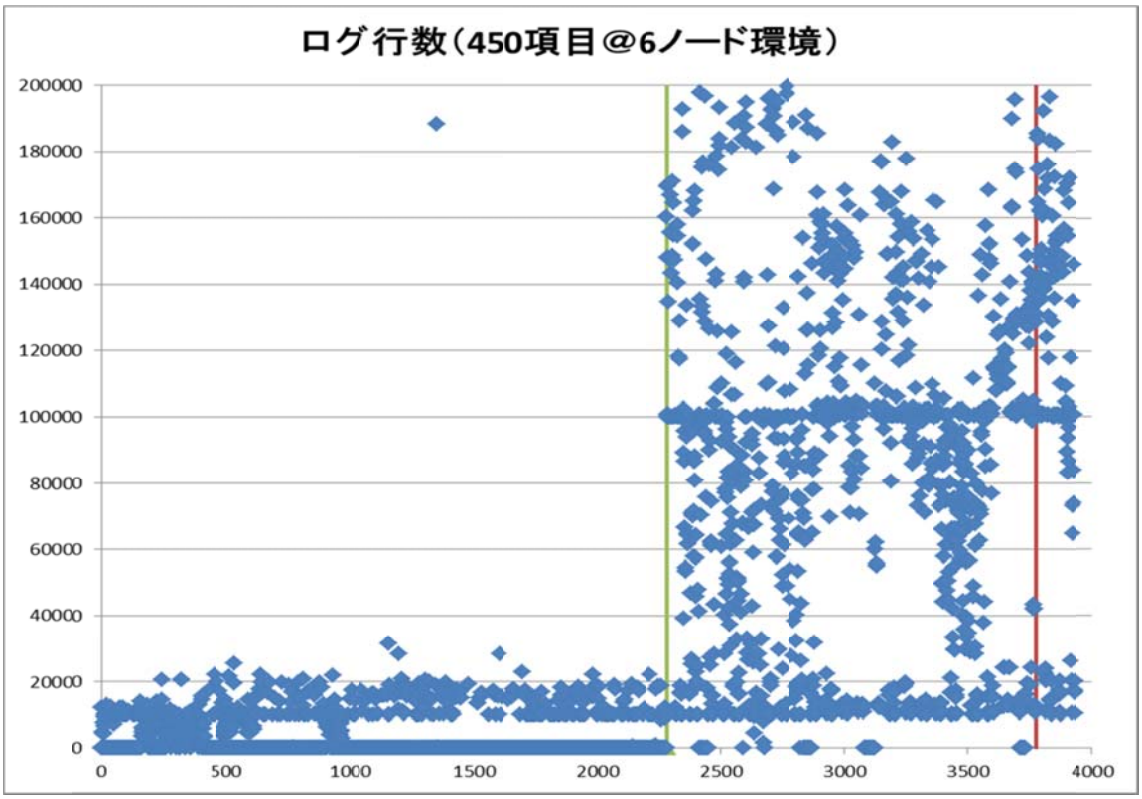
網掛は、ログレベル設定の目安を表す。（赤：デフォルトのファイルログレベル、青：デフォルトのメモリログレベル、緑：特徴ログの取得に必要なメモリログレベル）

上記の図が得られた結果です。紫色のマーク部分はデフォルトのログレベル、青色のマーク部分はデフォルトのメモリログレベルで、緑色のマーク部分が特徴ログを出力するためにメモリログレベルを変更する箇所です。

* 印箇所は、このログレベルのログが NM ログとして抽出されたことを表します。

** 印箇所は、このログレベルのログを特徴ログ (ON ログまたはログレベル 1 以下の NM ログ) として採用したことを表します。

次に、実際に各サブシステムのログレベルおよびメモリログレベルを上記の図のように設定した状態で、出力されるログの量を確認します。



上記の図は、この設定で実際に取得されたログの行数の分布です。

横軸はサンプルの通し番号で、時系列に並んでいます。縦軸は各サンプルのログ行数です。ログ行数は、テスト開始時点のログファイル行数と、ログアーカイブ取得時点のログファイル行数の差です。ログアーカイブ取得時点にメモリ上のログはログファイルにダンプされています。
ログ行数が 0 になっているところはメモリ上のログ領域不足によりメモリログの上書きが発生している箇所を表します。ログ領域不足の対策として、タテの緑色の線の時点で、ログ領域の大きさの設定を、デフォルトの Ceph ノード当たり 10,000 行から 100,000 行に拡張しています。
タテの緑色の線よりも右側でログ行数が 0 になっているところは、テスト中にログローテーションが行われる場合の考慮漏れによるログ回収ミスです。タテの赤色の線の時点で対策を実施しています。
この状態で、出力されるログの量（＝ログが出力される頻度）は、障害パターン毎に選択した特徴ログ（ONログまたはログレベル 1 以下の NM ログ）冒頭部分がメモリログ上で上書きされる前にログファイルに保存可能な範囲に収まったと考えられます。

6. まとめ

本稿では、Ceph を構成する各種ハードウェアで障害が発生してから復旧するまでの間の、OpenStack インスタンス (VM) 上で実行中のアプリケーション I/O への影響や、システム管理者が OpenStack 環境に対して行ったオペレーション (インスタンスの作成等) への影響を調査した結果を紹介しました。

また、ソフトウェア障害調査の場面で、調査対象となるソースコード範囲を狭めるために、Ceph の一連のログから、そのときに動作していた可能性のある処理ルートの候補をリストアップする仕組みや、フィールドサポートの場面で、事後にログから発生したハードウェア障害が何であったかを特定するために、ハードウェアの障害パターンに固有の特徴ログを抽出する試行について紹介しました。

※本書掲載内容の複写･無断転載を禁じます。

- 本書は 2017 年 8 月現在の情報に基づいて作成しております。
- VA Linux Systems Japan、VA Linux および VA Linux ロゴは VA Linux Systems Japan の商標または登録商標です。
- Linux は Linus Torvalds の米国およびその他の国における登録商標または商標です。
- その他、記載されている企業名、ブランド名および製品名は、各企業の商標または登録商標です。

VA Linux Systems Japan 株式会社
お問合せ: sales@valinux.co.jp
http://www.valinux.co.jp/