Vol.2016-OS-138 No.19
2016/8/9

OpenStackの挙動を リアルタイムに可視化するためのシステム

高橋 ひとみ[1,a)]　勝野 恭治[1,b)]　小野寺　民也[1,c)]

概要：Openstack とはオープンソースのクラウド管理ソフトウェア群であり、導入のしやすさ、構築環境に合わせたカスタマイズが容易にできることより使用ユーザが増加している。しかし各モジュールは分散協調し動作する構成を取っており、さらにリリースによっては大きく構成が変わるため、実際に導入、運用する中でトラブルが発生した場合、原因特定が難しい。そこで、本論文では各モジュールが出力するログに注目し、ログの解析によりリアルタイムに Openstack の挙動を監視するシステムを提案する。さらにこのシステムを既存のソフトウェアを用い実現し、インスタンスの作成処理を監視するツールを作成した。これにより各作成ステップ毎の処理時間をリアルタイムに表示が可能となり、管理者は Openstack の異常な挙動の検知およびその原因となるモジュールの特定が容易となる。

OpenStack Realtime Fine-grained Visualization System

Hitomi Takahashi[1,a)]　Yasuharu Katsuno[1,b)]　Tamiya Onodera[1,c)]

1. はじめに

Amazon EC2[1]、Microsofot Azure[2]、SoftLayer[3] など企業が商用のサービスとしてクラウド環境を提供する、パブリッククラウドは多く目にするようになった。一方、商用ではなくユーザの要求に即して柔軟にカスタマイズできるクラウド環境を構築するソフトウェアとして、オープンソースの OpenStack[4] が注目を集めている。Openstack の導入は自動インストールなどで簡単に構築できる [5] 一方、導入後に発生するトラブルの原因が複雑な構成のため特定できないという状況が発生する。Openstack のモジュール群は機能毎に存在し、かつそれぞれ独立で動作しており、これらのコンポーネント間は REST や RPC などの API で連携する構成となっている。そのためインスタンスの作成が出来ない、インスタンスの作成に時間が掛かる、コンピュータノードが認識されないなどの異常が発生した場合、原因の特定のためログを調査するがこれらのログは各モジュールからそれぞれ出力されるため、どのモジュールが原因となっているかを特定するのは困難である。

そこで本論文では既存の Openstack のコードを変更せず各モジュールが出力するログのみで、詳細な挙動をリアルタイムに監視するシステムを提案する。本論文では特に Openstack の主な機能であるインスタンスの作成に注目し、各モジュールが行うインスタンス作成のそれぞれの処理時間がリアルタイムに把握できる監視システムを実装した。ダッシュボードではインスタンス作成時にリアルタイムで処理時間が表示され、かつ、もしユーザが設定した閾値以上の処理時間がかかる場合にはユーザへアラートを通知することで異常がすぐに分かる。本論文では、監視システムを ElasticSearch[6]、Kibana[7]、Logstash[8]、Zabbix[9] を使用し、ログ解析プログラムのみを作成することで導入にはほとんどコストがかからず、Openstack の挙動が監視可能となる。

この後の本論文の構成として、まず第 2 節で本論文と同様に Openstack の実行状況を監視するツールを関連研究として紹介し、第 3 節で Openstack の概要について説明す

[1] 日本 IBM 株式会社　東京基礎研究所
IBN Japan, Ltd., 19-21, Nihonbashi, Hakozaki-cho, Chuo-ku, Tokyo 103–8501
[a)] hitomi@jp.ibm.com
[b)] katsuno@jp.ibm.com
[c)] tonodera@jp.ibm.com

る。次に第 4 節において設計方針を決定する予備実験の説明および本システムの設計を述べ、第 5 節で実際に本システムを構築し、Openstack への異常検出の有効性について説明する。最後に第 6 節にてまとめと今後の課題について述べる。

2. 関連研究

本システムと同様に Openstack の挙動を監視する既存システムおよびツールを紹介する。Sharma ら [10] はネットワークや RPC から Openstack を監視するシステム HANSEL を提案している。HANSEL では Openstack のモジュール間でやり取りされるメッセージを取得し、インスタンスの作成、削除などの動作の状態遷移を同定し、かつメッセージに含まれるエラーの内容を取得することで Openstack を監視する。このシステムでは明示的にエラーと出力されない場合や、処理に時間が掛かるといった Openstack の不安定な挙動は検知できない。

Stacktach[11] は Rackspace 社が開発した Nova の挙動を監視するツールである。このツールは RabittMQ を監視し Nova 間や Nova 以外のモジュールに送られるイベントや通知メッセージを収集し Nova の挙動を監視する。このツールはモジュール間のメッセージを監視するツールであり、メッセージが送信されない、同一モジュール内の詳細な挙動までは把握できない。

Openstack のプロジェクトの一つである OsProfiler[12] は Python のライブラリであり、Openstack の性能プロファイリングを可能とする。このライブラリは Openstack の一つのサービスが実行された際、どの API コールが呼ばれているか、その API が実行終了するまでにどのくらいの時間が掛かったかといった性能を詳細に取得できる。このツールの用途は何か異常が発生した場合やサービスのオーバヘッドがどの部分にあるかを特定するためのツールであり、Openstack の挙動を常時監視するためには使用されない。

最後に Openstack の運用モニタリングツールとしてよく使用されている ElasticSearch、Logstash、Kibana を紹介する。ElasticSearch は検索エンジン、Logstash はログの収集とインデックス、Kibana はユーザへのダッシュボート機能を提供しており、これらのソフトウェアを組み合わせて監視システムとして使用される。ただし、これらの 3 つのソフトウェアを使用してもログを解析する部分がないため、本論文が提唱するログの解析モジュールが別途必要となる。

3. Openstack

Openstack とは 2010 年に発表され、現在、なお開発が進められているオープンソースのクラウド管理機構である。管理者やユーザが、クラウドの基本管理要素であるコンピュータ、ストレージ、ネットワーク資源の操作を GUI のダッシュボードから操作できる Web インタフェースも備えている。Openstack には多くの関連するプロジェクトが存在しており、アプリケーションの自動インストールやインテグレーションを行う Heat、メータリングを行う Celiometer なども存在する。Openstack の機能はコンポーネント毎に実現されており、これらのコンポーネント群は独立して動作し、他のコンポーネントおよびユーザからのリクエストを受けて必要な処理を行っていく分散型のアーキテクチャとなっている。そのため、全ての挙動を把握するマスターとなるコンポーネントは存在しない。各コンポーネント間のデータのやり取りは REST API もしくは RPC により実現されており、RPC にはスケーラビリティを考慮し AMQP (Advanced Message Queueing Protocol) [13] が使用されている。

3.1 コンポーネント

Openstack の主要なコンポーネントを図 1 に示す。各コンポーネントの機能は以下のとおり。

- Horizon：ユーザおよび管理者が利用する Web GUI インタフェースのダッシュボード。ユーザは Horizon 以外にも CUI による操作が可能である。
- Nova：インスタンスの作成や管理を行うモジュール。クラウドを管理するホスト（コントローラノード）に存在する Nova および、実際にインスタンスを作成するホスト（コンピュータノード）に存在する Nova Compute の 2 つが存在する。ユーザからの要求は REST API を通じ Nova が受け付け、Nova Compute へインスタンスの作成要求を発行する。これらのインスタンスの情報などはすべて Nova が所有する Nova Database に格納される。
- Keystone：主にユーザ認証を行う機構であり、ユーザやプロジェクト毎にどのサービスが使用可能かなどの情報が格納されているサービスカタログ機能も提供する。
- Cinder：インスタンスに対しブロックストレージの提供を行う。インスタンスのスナップショットの保存にもこのストレージが使用される。
- Glance：VM イメージの管理を行う。イメージの登録、取得などの機能を有する。
- Neutron：ネットワークのサービスを提供する。REST API から受け付けた要求によりネットワークやルータの作成、インスタンスへのネットワーク割当などを行う。様々なネットワークの機能をプラグインとして追加でき、各プラグインに存在するエージェントがコンピュータノード上のネットワークを設定する。

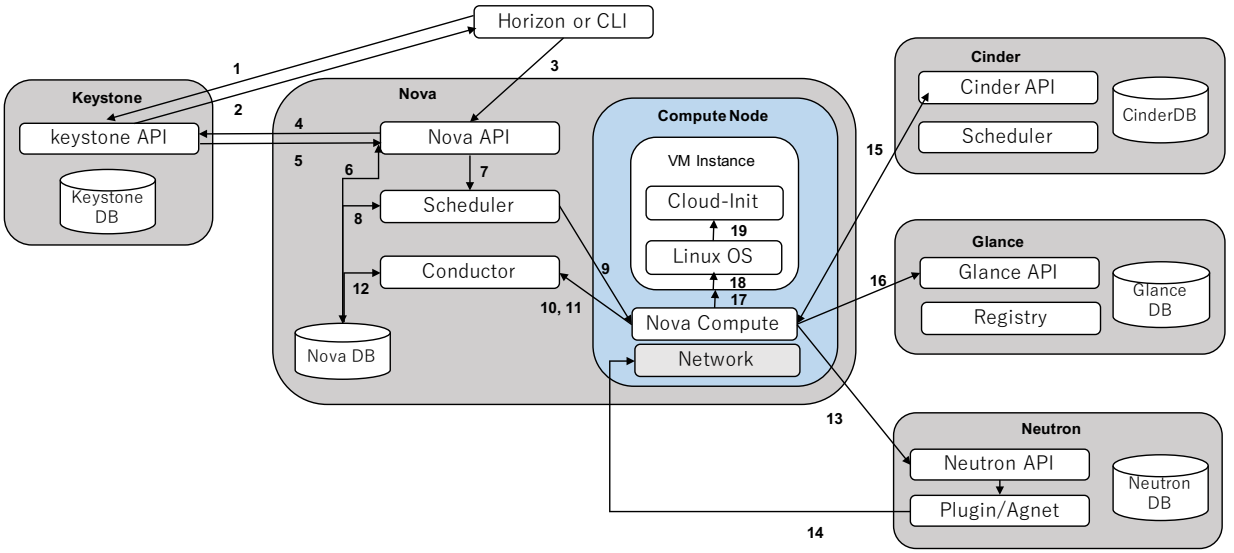

図 1 Openstack のアーキテクチャ

3.2 インスタンスの作成フロー

本小節では Opensatck の機能の内、ユーザが頻繁に使用する機能として、インスタンスの作成フローと処理の説明を図 1 における番号とともに説明する。

(1) Horizon もしくは CLI のコマンドから、ユーザの認証を行うため、Keystone へユーザ名やパスワードの認証情報が送信される。
(2) Keystone はユーザが所属するプロジェクトやロールの情報とともにトークンを返答する。
(3) Horizon もしくは CLI はトークンとともに Nova API に対し、インスタンス作成のリクエストを送信する。
(4) Nova API は送信されたトークンが正しいか、操作権限があるかを Keystone に問い合わせる
(5) Keystone は各 API から問い合わせたトークンが正しいかどうかの結果を返答する
(6) Nova API はリクエストが正しいものかを検証し、Nova DB よりインスタンス作成に必要な情報を得ると同時に、これから作成するインスタンスの初期情報のエントリを作成する。
(7) インスタンス作成要求が AMQP を通じスケジューラーに送信される。
(8) スケジューラはどのコンピュートノードに対し、要求されたインスタンスを作成するか決定する機能を持つ。そのため、現状のコンピュータ資源の情報を Nova DB より取得しコンピュータノードを決定する。
(9) スケジューラは AMQP を通じ、インスタンスが作成されるコンピュートノードに対しインスタンス作成の要求を送信する
(10) Nova Compute が Conductor に対し、作成するインスタンス情報を取得するため要求を送信する。
(11) Nova Conductor は AMQP より要求を取得する
(12) Nova Conductor は Nova DB からインスタンスを構築するために必要な情報を取得し、Nova Compute へ送信する。
(13) Nova Compute は Neutron に対しインスタンスへのネットワーク作成や設定を行うよう要求を送信する。
(14) 要求を受信した Neutron は IP の割当やゲートウェイなどの設定をコンピュータノードに対して行う。
(15) Nova Compute はユーザの要求があればブロックストレージをインスタンスに割り当てるため、Cinder に対し要求を送信する。
(16) Nova Compute はユーザが指定した VM のイメージを Glance より取得する。
(17) Nova Compute は指定されたイメージおよびスペックで Hypervisor を起動しインスタンスを作成する。
(18) 作成された VM 上で OS が起動する。
(19) Cloud Init が起動しネットワークや SSH 鍵を設定する。

3.3 ログ形式

Openstack の各インスタンスのモジュールはイベントが発生するとユーザが指定したログレベルに応じたログを生成する。ログレベルを Debug レベルに設定した場合、以下のようなログが出力される。”2015-10-23 04:11:03.803 13272 DEBUG oslo_concurrency.lockutils [req-465134df-1918-43af-84dc-6429bf106ad4 3de18b5c54d946a09e763fe2ef9a3ae6 a179db4e10764ea281d0b07dc2ff9412 - - -] Releasing semaphore "neutron_admin_auth_token_lock" lock /usr/lib/python2.7/dist-packages /oslo_concurrency/lockutils.py:404”。この メッセージは”イベント発生時間 プロセス ID ログレベル モ

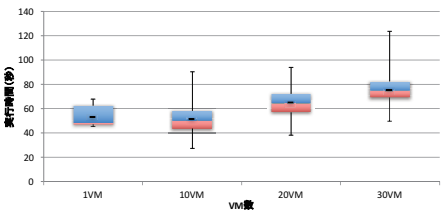

図 2 Nova の処理時間

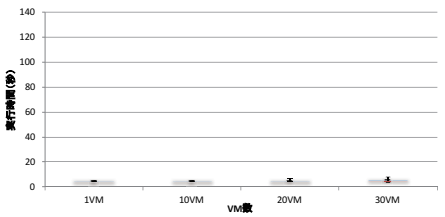

図 3 OS の起動時間

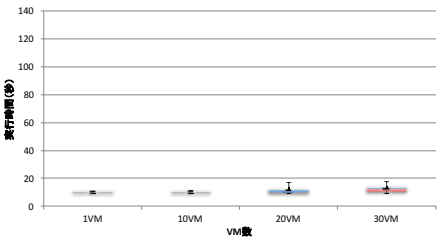

図 4 Cloud Init の処理時間

ジュール名 [リクエスト ID ユーザ テナント - - -] デバッグメッセージ コード:該当行”の内容が含まれている。リクエスト ID とはモジュールが何かの処理のリクエストを受信し、処理が始まる時に生成される ID である。このリクエスト ID は該当するリクエストの処理に関連するログメッセージに追加され出力されるが、モジュール間では引き継がれないため、モジュールをまたぐとリクエスト ID が変更されてしまう。

4. 設計

本節では Openstck の監視システムの設計について述べる。本監視システムの目的はログから Openstack の挙動を把握することである。本論文ではユーザが頻繁に使用する機能であるインスタンス作成の監視に焦点を絞りシステムの設計を行った。

Openstack では第 3.2 節で述べたようにインスタンス作成のステップは、ユーザからの要求を受けユーザがインスタンスへログイン出来る状態となるまで、大きく分けて、Nova のプロセス、VM 作成時の OS の起動プロセス、OS 起動後に実行される Cloud Init のプロセスに分かれる。監視システムでは、インスタンスの処理時間の異常を見つけるために、インスタンス作成における処理時間の内訳を調査し、最も大きな割合である処理を詳細に監視する必要がある。そこで Nova、OS の起動、Cloud Init それぞれの処理時間を実環境で測定した。

4.1 予備実験

予備実験環境として 13 台のコンピュータノードおよび 1 台のコントローラノードに Openstack kilo をインストールしクラウド環境を構築した。各ホストのハードウェアは CPU 8 Core　デュアルソケット、RAM 128GB、HDD 1TB である。この Openstack 環境にインスタンスを同時にそれぞれ 1、10、20、30 個と Ubuntu15.04 のイメージを用い作成し、Nova の処理時間、OS の起動時間、Cloud Init の処理時間を 100 回計測した。

図 2、3、4 に Nova の処理時間、OS の起動時間、Cloud Init の処理時間をそれぞれ示す。X 軸は同時に作成したインスタンスの個数、Y 軸は処理時間（秒）となっており、100 回計測の最大値、中央値、最小値、パーセンタイルを示している。この結果では Nova の処理時間に多くの割合が取られており、同時に作成するインスタンスの個数に大きく影響を受けている事がわかる。1 および 30 インスタンスの結果を比較すると、Nova の処理時間は中央値、分散ともに同時インスタンスを作成する個数が大きいほど処理時間が長くなる。このことより、Nova でのインスタンス作成の処理を詳細に見ていくことで、作成処理の異常を検知できる可能性が高いと分かる。そこで本監視システムは Nova の処理に注目する。

4.2 監視システムの全体構成および動作フロー

図 5 に本監視システムの全体構成および動作フローを示す。本システムはログ収集、一時的なログデータの貯蓄、日付やログレベルなどのログの基本解析、詳細なログの解析エンジン、データ検索のための DB、ダッシュボート、異常時のユーザ通知のモジュールからなる。本システム全体のフローは以下となる。

(1) コントローラ、コンピュータノードで稼働している Openstack のモジュールは各ホストにログを出力する。
(2) 各コンピュータノードにログ収集のエージェントを導入し、Openstack から出力されるログを本監視システムを導入したホストへ送信し、一時保存用の DB に格納する。
(3) 監視システム上で稼働しているログ基本解析モジュールは一時保存の DB よりデータを取り出し、基本的な情報をタグ付けし、検索エンジン用 DB に送信する。
(4) ログ解析エンジンは定期的に検索エンジン用 DB より情報を取り出し、インスタンス作成時のログ抽出やタグ付け、プロセスごとのセグメントを行い、DB に再び格納する。
(5) ユーザは可視化モジュールより、ログの表示や各グラフから Openstack の状態をチェックする。
(6) 通知モジュールはユーザが定めた閾値以上の時間が掛かる場合、異常を通知する。

4.3 ログの解析手法

Openstack の管理機構は第 3.2 節で述べたように多くのモジュールを介し動作し、各モジュールはイベントが発生するとログのレベルにより様々なメッセージを出力する。

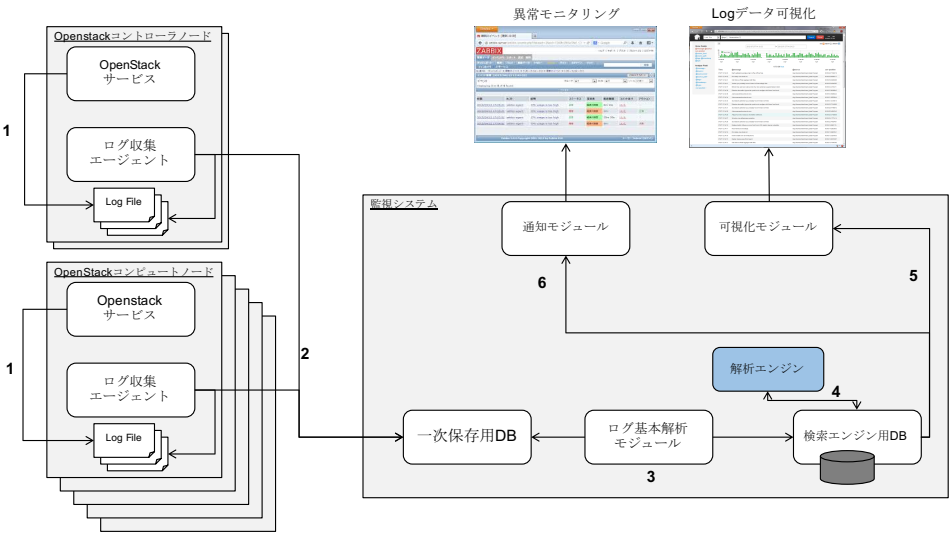

図 5 監視システムのアーキテクチャ

本論文ではインスタンス作成に焦点を当てるため、作成に関連するログのみを解析する必要がある。さらに複数同時にインスタンス作成要求を受信した際には、それらのインスタンスが入り混じったログが出力されるため、取得したいインスタンスの作成フローに関するログのみを抽出する必要がある。

Openstack には各モジュールでログのレベルを WARN、INFO、DEBUG などに設定でき、ログレベルに応じたメッセージを出力する。本システムでは通常状態の処理時間も監視するため、全モジュールのログレベルは DEBUG レベルでの運用を行う。また、解析に必要なモジュール群は同一ホスト上に存在するとは限らない。コントロールノードとコンピュータノード、もしくはコントロールノードが複数存在する運用も考えられるため、全ホストに NTP を運用させホスト間の時間が同期されていることを前提とする。

全ホスト上で時間が同期されていた場合、全てのモジュールからログを収集し時間でソートするとインスタンス作成の際に行った処理の間に出力されるログを取得できるが、作成時間の間のログのみを取得しても対象となるインスタンス作成のメッセージのみとは限らない。インスタンス作成処理の他にもすでに存在するインスタンスの管理や、対象となるインスタンスより前に作成要求が送信され、まだ作成が終えていないインスタンスの処理なども存在するため、ソートしたログより対象となるインスタンスのみに関連があるログを抽出する必要がある。そこで本ツールはインスタンス ID とリクエスト ID より、関連あるインスタンス作成のログを抽出する。

インスタンス ID とは Openstack が自動的に割り振る一意の識別子である。本システムはインスタンス作成要求の時間、インスタンス作成終了の時間をログより取得し、その間の監視対象となるインスタンスの ID が出力されているログを抽出することで、対象のログをおおまかに抽出できる。しかし、このインスタンス ID はインスタンス作成に関連があるメッセージ全てのログに表示されるわけではなく、また、インスタンス作成の初期ですぐに割当られるわけではない。さらに、インスタンス作成とは関連がない処理にもインスタンス ID が表示されてしまう。この関連がない処理とは定期的に起動されるタスクであり、インスタンスの状態やコンピュータリソースなどの監視を行うタスクである。このタスクはモジュールの起動から同一のリクエスト ID を使用するため、一度、定期タスクのリクエスト ID が判明してしまえば、そのタスクが出力するログを除去できる。そこで本ツールは以下の処理を順に行い、インスタンス作成に関連したリクエスト ID、作成されたインスタンス ID を含むログを抽出する。具体的にはインスタンス ID のみを含むログ、リクエスト ID のみを含むログ、両 ID を含むログとなる。

(1) 定期タスクのみに表示されるメッセージからそれに紐づくリクエスト ID を取得する
(2) インスタンス作成時に出力されるメッセージから、作成されたインスタンス ID を取得する
(3) (2) で取得したインスタンス ID が割り当てられているインスタンスにおいて、作成終了までに出力されたログを取得する
(4) (3) で取得したログにおいて、該当インスタンス ID が出力されているメッセージから全リクエスト ID を取得する
(5) (4) で取得したリクエスト ID をログより抽出する
(6) (5) で取得したログから (1) で取得したリクエスト ID に紐づくメッセージを削除する
(7) 抽出したログへインスタンス ID のタグを追加する

以上の動作により、インスタンス作成に関連があるログメッセージのみの抽出およびタグ付けが可能となる。

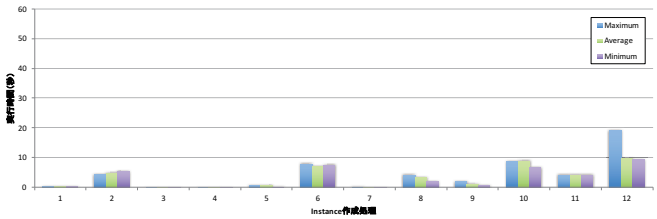

図 6 正常状態でのインスタンス作成状況の処理時間

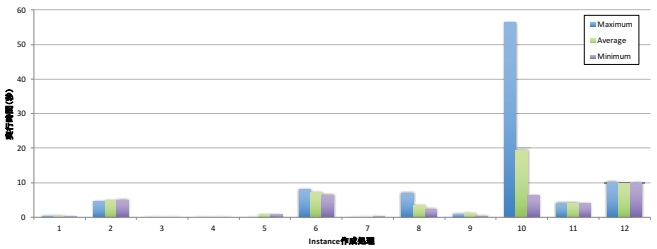

図 7 異常状態でのインスタンス作成状況の処理時間

4.4 表示方法

前小節より取得できたログメッセージの解析手法より、インスタンス作成時の挙動とそれに掛かる処理時間が取得できるようになった。しかし、ログは DEBUG レベルのメッセージのため非常に内容が細かく、ユーザが一見して Openstack の状態が異常かどうかを判断できない。そこで、本監視システムは Nova の作成時のプロセスを 10 プロセスに分け、それぞれの処理に関連のあるメッセージを分類し、各処理の時間を算出するプログラムを作成した。さらに、本システムではこの 10 段階のプロセスとともに、インスタンス作成後の OS の起動および Cloud Init の処理時間を加え、計 12 プロセスに分割する。各プロセスと図 1 におけるフロー番号の対応を表 1 に示す。

表 1 プロセスとフロー番号の対応

プロセス番号	図 1 におけるフロー番号
1	(1) (2) (3) (4) (5)
2	(6)
3	(7)
4	(8) (9)
5	(10)
6	(11)
7	(12)
8	(13) (14)
9	(15)
10	(16) (17)
11	(18)
12	(19)

5. 本システムの実装および有効性の確認

各モジュールにおける DEBUG レベルのログを収集し、収集したログよりインスタンス作成に関連があるログの抽出、各プロセス毎のログを分類し処理時間を測定した。

その結果を図 6、7 に示す。X 軸は第 4.4 節で述べたプロセス、Y 軸は各プロセスの処理時間を秒で示している。また、図 6 は Openstack の正常時におけるインスタンス作成時間、図 7 は Openstack 異常時におけるインスタンスの作成時間を示している。図 7 では、図 6 と比較し、プロセス 10 の時間が平均 11 秒も異なり、特に最大値では 46 秒も異なる。プロセス 10 は Nova Compute が各ハイパーバイザに対し KVM の作成を行うプロセスであるため、Nova Compute の異常、もしくは各コンピュータノードのハイパーバイザが異常であると判別がつく。

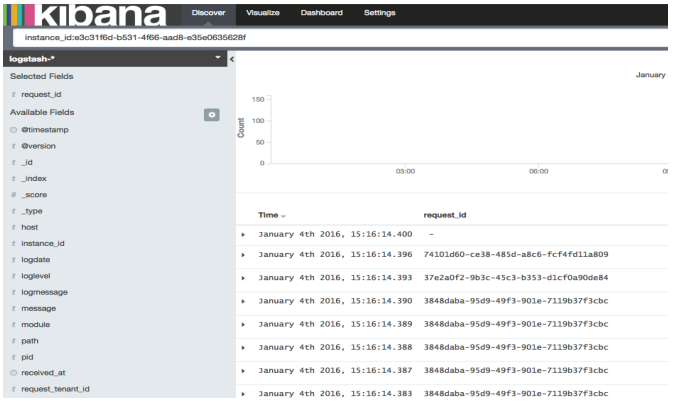

図 8 Kibana によるログの表示

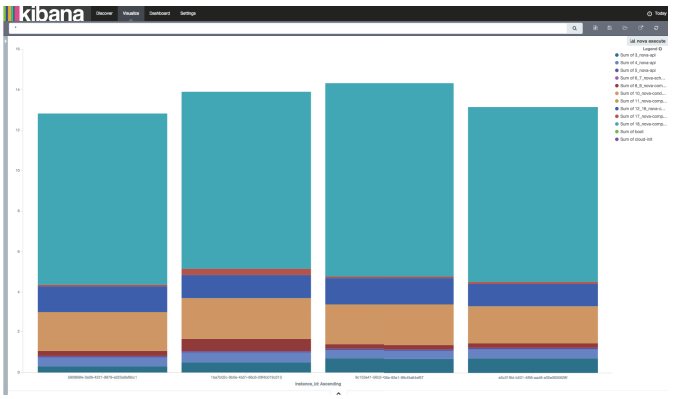

図 9 Kibana によるインスタンス作成時間のグラフ表示

5.1 Openstack における監視システムの構築

本監視システムを全て自動化するため、ログ解析エンジン以外の部分は既存のソフトウェアを用い、本システムを構築した。図 5 に示されている、ログ収集エージェントおよびログ基本解析モジュールには Logstash、一次保存用 DB には Redis DB[14]、検索エンジン用 DB には Elasticsearch、可視化モジュールには Kibana、通知モジュールには Zabbix[9] を採用し、本システムを構築した。これによりユーザは Kibana を通しインスタンス作成における異常の原因や、Zabbix による異常検知が可能となり、Openstack の異常の兆候を簡単に発見できるようになる。

5.2 Kibana による表示

本モニタリングツールを実際に Openstack 上の VM に構築し、Openstack の挙動を監視した。図 8、9 にログの表示例、インスタンス作成時の時間を示したグラフの表示例を示す。

図 8 では本モニタリングで作成したツールにより Instance_id のタグが追加されている。このタグによりユーザは調査したいインスタンス ID のタグを検索すれば、このインスタンスの作成に関連するログを取得できる。図 9 はインスタンス作成時間のグラフをリアルタイムで表示をしている。横軸はインスタンス ID、グラフ内の色は Nova のそれぞれのプロセスを示している。本解析ツールでは、ログにインスタンス ID のタグを追加、プロセス毎にログを分類し、各プロセスの処理時間を算出している。その解析結果を再び Elasticsearch に書き込む事で、ユーザはリアルタイムにインスタンスの作成時間を取得できる。このように Kibana のダッシュボートを使用することで、Openstack におけるインスタンス作成時間がリアルタイムに表示でき、インスタンス作成時における異常検知が容易になった。

6. まとめと今後の課題

本論文ではログより Openstack の挙動を監視するシステムを提案した。今回はインスタンス作成に注目し、ログを詳細に解析する独自のプログラムは作成したが、ログの収集、データの検索、データの表示などは既存のプログラムを用い、簡単にかつ迅速に構築できる監視システムを構築した。これにより管理者は Openstack においてインスタンス作成時の異常を Nova の詳細なプロセスの処理時間から推定できる。

今後の課題として、現状のシステムはユーザが定めた閾値以上の処理時間が経過した場合にのみ異常を通知するだけだが、異常に達するまでの予兆をシステムが自動的に検知し、ユーザに何をすべきか提案するようシステムを改善する。また、現状では全モジュールを DEBUG レベルに設定しているがログ出力の負荷が高くなってしまうため、通常は WARN などのログレベルで運用を行い、異常の予兆を検知した際に DEBUG のログレベルへ動的に変更する機能を追加したい。

参考文献

[1] Amazon EC2: https://aws.amazon.com.
[2] Microsoft Azure: https://azure.microsoft.com.
[3] SoftLayer: http://www.softlayer.com.
[4] Openstack: https://www.openstack.org/.
[5] Sobeslav, V. and Komarek, A.: *Proceedings of the 4th International Conference on Computer Engineering and Networks: CENet2014*, chapter OpenSource Automation in Cloud Computing, pp. 805–812, Springer International Publishing (2015).
[6] Elasticsearch: https://www.elastic.co/jp/products/elasticsearch.
[7] Kibana: https://www.elastic.co/products/kibana.
[8] Logstash: https://www.elastic.co/products/logstash.
[9] Zabbix: http://www.zabbix.com/.
[10] Sharma, D., Poddar, R., Mahajan, K., Dhawan, M. and Mann, V.: HANSEL: Diagnosing Faults in OpenStack, *Proceedings of CoNEXT'15* (2015).
[11] StacktTach: http://stacktach.com/index.html.
[12] OsProfiler: https://github.com/openstack/osprofiler.
[13] Vinoski, S.: Advanced Message Queuing Protocol, *IEEE Internet Computing*, Vol. 10, No. 6, pp. 87–89 (2006).
[14] RedisDB: http://search.cpan.org/dist/RedisDB/.