応募区分：提言型論文

OpenStack クラウド基盤における従量課金の価値と実装検討

斎藤　彰宏（さいとう　あきひろ）
日本アイ・ビー・エム株式会社 ソリューション
事業部　ハイブリッドクラウドSC

倉前　裕成（くらまえ　ひろあき）
日本アイ・ビー・エム株式会社 ソリューション
事業部　ハイブリッドクラウドSC

■ 要約

クラウド市場は年率約 50%の著しい成長を続けており，クラウド事業者のサーバ数は顕著な増加傾向にあるが， 採用サーバは ODM サーバ中心にシフトしつつある．クラウド事業は高成長，高利益の有望分野だが，巨額の設備投資は経営上の大きな負担で，より安価な ODM サーバを選択する原因になっている．クラウド事業者にメーカサーバを訴求するためには従量課金が有効であり，従量課金測定機能（コレクタ）の技術実装としては OpenStack が最も適している．概念実証として OpenStack からサーバ稼働情報を取得するモジュールを作成し動作を確認した．また投資採算性シミュレーションでは， ODM サーバに対する明確な優位点として，投資回収期間が短く，事業非成長時の損失を最小限に抑えられる点も示された．また実際にサーバ従量課金を導入いただいたクラウド事業のお客様からの評価コメントでもその価値は裏付けられている．

目次

1. はじめに

クラウド市場は年率約 50%の著しい成長を続けている[1]．市場調査では 8 割の企業において新規システムはクラウド採用を検討すると回答しており[2]，企業 IT 基盤の「クラウドファースト」へのシフトは明らかである．またクラウド市場の成長でサーバ機器市場にも大きな変化が起きている．これまでサーバ機器の主たる顧客は企業や官公庁であったが，ごく近い将来にクラウドなどデータセンター事業者が企業オンプレミスを稼働サーバ台数で上回るのは確実視されており[3]，今後のサーバ機器の主たる顧客は図 1 のようにクラウド事業者になると考えられる．一方でクラウド事業者の稼働サーバにおける ODM サーバ (Original Design Manufacturer Server) の割合は，2009 年には 28%だったが，2014 年に約 50%に達した[4][5]．つまり一般企業における「クラウドファースト」が進む一方で，クラウド事業者における「ODM ファースト」も進行している状況である．サーバメーカはクラウド事業者が購買を決定する基準(Buying behavior)を把握し，その基準において ODM サーバよりも上回る価値があることを訴求できるように価値訴求を見直す必要がある．本稿ではクラウド事業者がメーカサーバより，ODM サーバを優先して採用するようになった原因を分析した上で，クラウド事業者が必要とするオファリングとその実装について検討を行い，OpenStack 環境に対応したハイブリッド従量課金がクラウド事業者にニーズに応えることを採算性評価のシミュレーションで検証する．なお，本稿は筆者の個人的な見解であり，所属する団体，組織の意見を代弁するものではない．

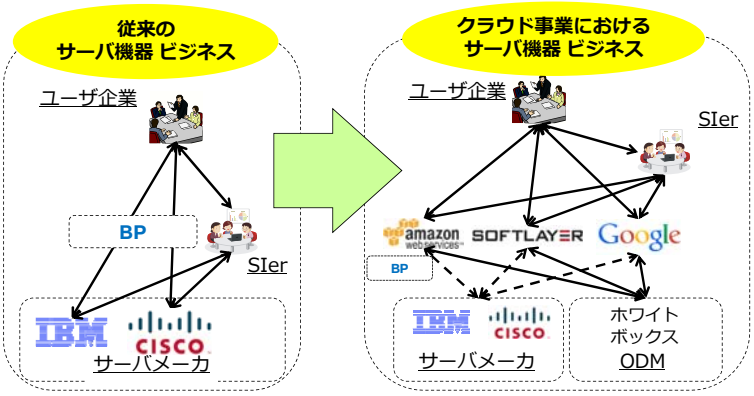

図 1 サーバ機器ビジネスの環境変化

2. サーバ調達の従量課金コンセプト

クラウド事業者が ODM サーバを選択している購買基準が低価格性であることは，複数の調査で共通して指摘されている[4][5]．メーカサーバとの価格差は，Morgan Stanley Research の調査[4]で約 40%，IDC の市場調査[5]において約 45%であり，メーカサーバと ODM サーバの価格差は約 40～45%と推定される． ODM サーバの品質や保守に不満を感じているクラウド事業者も多いが[5]，それでも低価格をサーバの絶対的な購買基準とする『Value for Money』がクラウド事業者の Buying behavior における注目すべき特徴である．

2.1. クラウド事業者の財務傾向

Amazon AWS などクラウド事業の決算を公表している大手クラウド事業者の近年の財務諸表[8][9][10]では，クラウド事業の営業利益率(Operating Income Margin)は 20%近く，極めて高い利益水準である(図 2) ．現在クラウド事業者間での価格競争は激化しているが，依然としてクラウド事業が，高成長かつ高利益の良質なストックビジネスであることが読み取れる．

	Amazon	Google	Rackspace
売上成長率	49.0%	10.3%	16.9%
営業利益率	16.9%	25.6%	13.2%

図 2 大手クラウド事業者 成長率，利益率(2014, 2015)

一方でクラウド事業者の固定資産回転率(Fixed Asset Utilization)の平均は 1.8 回であり，大手 IT 企業の平均である約 8.0 回と比較して明らかに低い．クラウド事業は高い利益を産み出すが，前提として莫大な設備投資が必要であり，巨大な設備投資を，規模の経済を追求することによって回収するビジネスモデルであることは明白である．このようなビジネスモデルでは，事業環境の変化や資金調達の滞りで，短期間で資金ショートが発生する．実際に，クラウド事業の先駆けと言える世界最大のデータセンター事業者であった Exodus Communications 社(2001 年破産)[12]や，クラウドストレージ事業大手の Nirvanix 社 (2013 年破産)[13]は，両社とも事業は成長していたにも係わらず，キャッシュフロー悪化による運転資金不足でごく短期間で倒産している．裏付けのため現在，世界最大のクラウド事業専業事業者である Rackspace 社の財務諸表[10]を分析したが同社の売り上げを 2014 年から 1%を成長させてもフリーキャッシュフローは 1.6 億円増だが，固定資産を 1%削減すればフリーキャッシュフローは 14.3 億円増となり，より大きな財務メリットが生じる．この傾向は大手クラウド事業者でほぼ共通していると考えられ，設備投資の負担軽減はデータセンター事業者にとって極めて重要な経営上の課題であると言える．

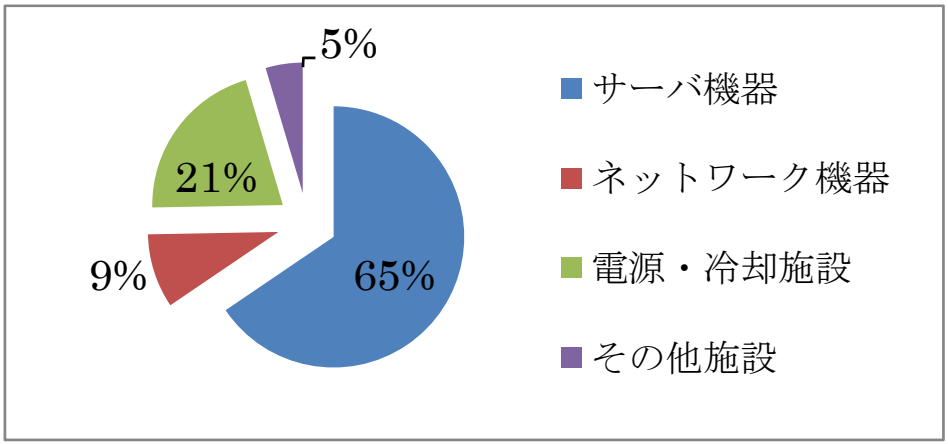

図 3 Amazon AWS のクラウド基盤費用割合[11]

また Amazon AWS はクラウド基盤の費用割合を公開しているが(図 3), ストレージをサーバ(分散ストレージ技術)で構成していることもあり突出して高い設備コスト(65%)がサーバ調達費用

に集中している．すなわち前述のクラウド事業者の重大な経営問題である設備投資負担の解決にはサーバ費用の削減が最も効果的であることに疑いの余地はない．クラウド事業者における「ODM ファースト」を覆すには，サーバメーカは，徹底してクラウド事業におけるサーバ調達の財務上のメリットを追求する必要がある．一般企業を対象とした過去の調査[14]では，サーバの購買決定における基準として「仕様（機能，性能，拡張性，品質)」が「コスト」を上回っていたが，クラウド事業者における購買決定基準は，この点において一般企業とは異なると言える．

2.2. クラウド事業者におけるサーバ調達契約

クラウド事業者がサーバを調達する際の契約方法はODM サーバとメーカサーバで異なっている．ODM サーバは生産委託の仕組み上，必然的に一括支払いでの買い取りが主となるが，メーカサーバの場合，クラウド事業者の調達はファイナンス・リース契約で調達されるのが主である[4]．急激な資金流出を防ぐキャッシュフローの観点からは，支払いが分割されるリース契約に利点が多いが，総支出額はサーバ機器単価が安い ODM サーバに利点が多い．従って資金調達に余裕がある大手クラウド事業者では特に ODM サーバを優先して選択していると考えられる．

3. サーバ調達の従量課金コンセプト

ODM サーバとメーカサーバの調達は，一括購入とリースでの分割支払いで，契約方式の違いはあるが，どちらも支払い総額が固定であり，図 4 のようにクラウド事業の事業収益の変動とは，連動していない点は共通している．

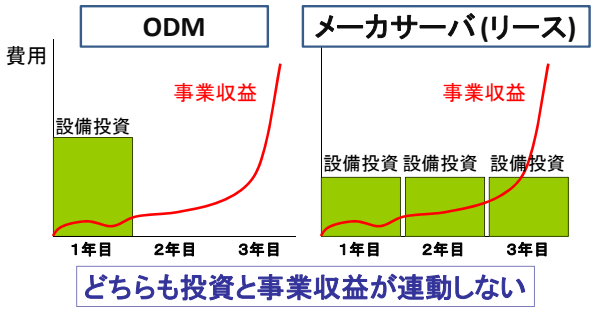

図 4 従来の設備投資とクラウド事業収益

クラウド事業はユーザ数と利用量が増えることで収益増加を計る従量課金のビジネスであるため，事業収益と設備投資が連動しないことで生じるキャッシュフローのギャップは，そのままクラウド事業者の財務リスクとなっている． 言い換えればクラウド事業者は不確実な将来の事業収益予想を根拠に設備投資額を決定せざるを得ないため，設備投資は多分に投機的な決定とならざるを得ない．この経営問題の解決する方法としてサーバ従量課金を検討した．従量課金の実装上の特徴は 2 点で，１点目は解約・再契約可能なオペレーティング・リース契約の活用で，柔軟な契約変更を可能とすること．2 点目は料金を固定費用から従量課金に変更し，図 5 のようにクラウド事業の収益と設備投資を連動させることにより，クラウド事業者の財務課題の抜本的解

決に貢献できるため，サーバ訴求力向上が期待できる．

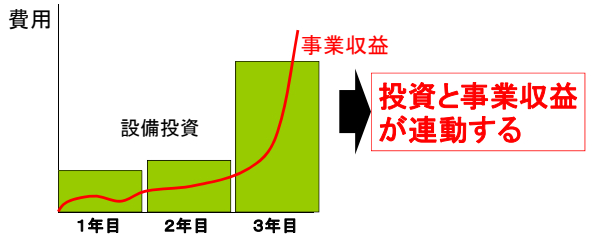

図 5 従量課金の設備投資とクラウド事業収益

ただしこの従量課金には，サーバメーカ採算性に致命的な欠陥がある．サーバの粗利益率を30%と仮定した場合，同条件での従量課金では課金対象時間が 70%以下で原価割れが発生する．サーバ資源利用率が 70%以上の事業者は半数以下であり，大部分のビジネスでサーバメーカは損失を被ると推測されるため，現実には完全従量課金はサーバビジネスとして成立しないと考える．

3.1. サーバ利用状況測定機能実装方式

完全従量課金の改良としてハイブリッド従量課金を検討した． ハイブリッド従量課金では図6 のように，サーバ機器の半分を固定課金，残り半分を従量課金で提供する形態になる．クラウド事業者のサーバ利用率が導入サーバの 50%に達するまでは，固定費用となるが，利用率が 50%を越えた時点からは，事業収益の変動と連動して設備投資額を変動が可能になる．クラウド事業者にとっては，サーバ設備投資費用を最大 50%コスト削減可能である上に，リースによる分割払いでキャッシュフローの負担を軽減できる点や，使用量後払いのため日々の事業状況に基づいて，定量的な財務的判断として設備投資額の決定が可能となる点により，ODM サーバを上回る財務メリットを享受できる．メーカにとっても納入サーバの 50%の売り上げはストックビジネスとして確保できるため，原価割れの危険性を低減できるメリットがある．

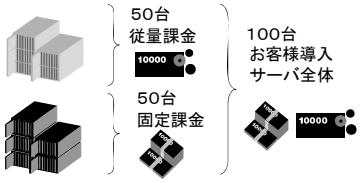

図 6 従量課金オファリングの基本コンセプト

3.2. 従量課金機能の実装モデル

ハイブリッド従量課金のアイデアを実現する実装検討を実施した．まず従量課金の実装構造モデルは，図 7 に示す通り，『従量課金対象サーバ』『管理サーバ』『データ管理 PC』の３つのコンポーネントで構成することとした．

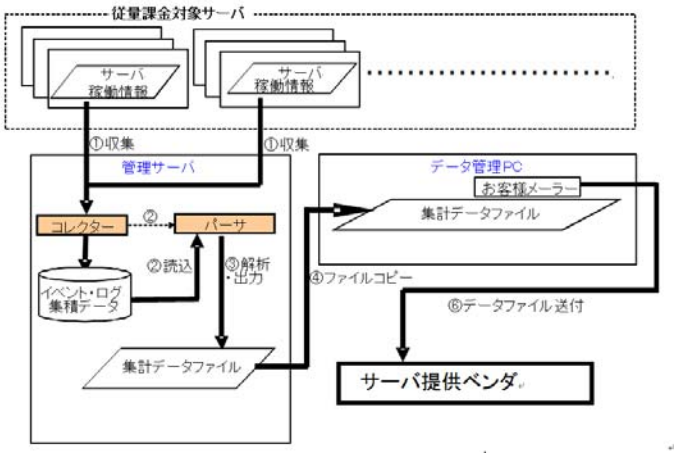

図 7 サーバ利用状況測定機能の構造

① 従量課金対象サーバ

従量課金対象サーバは従量課金対象であり，利用状況の取得と蓄積を行う対象である．

② 管理サーバ

従量課金対象サーバの利用情報を収集する機能(コレクタ)，整形を行う機能(パーサ)から構成される．コレクタは，一定時間毎に対象サーバのから利用情報を収集して，イベント・ログ集積 DB に蓄積を行う．『パーサ』はイベント・ログ集積 DB を読み込み，集計データの製表を行いメーカに送付する『集計データファイル』を生成する．

③ データ管理 PC

データ管理 PC は，管理サーバで生成された『集計データファイル』の閲覧と同ファイルをサーバ提供ベンダに送付する役割である．

4. 従量課金測定機能（コレクタ）の技術実装検討

課金対象サーバの利用情報を収集する機能(コレクタ)は従量課金の実現の上で必須となる機能である， 従量課金の測定対象はサーバの通電時間（電源 ON/OFF)， OS の起動時間， CPU 使用量， メモリ使用量， ディスク使用量等が考えられる．代表的な実装候補となる以下４方式について実装の検討を行う．

①測定エージェントによる計測
②監視サーバからの死活監視で計測
③各サーバの OS の機能で計測
④OpenStack の機能で計測

4.1. 計測エージェントソフトで利用時間計測

従量課金対象サーバ上に測定エージェントを導入し，資源利用状態を監視，ローカルに一時保存しておき，監視サーバで定期的に情報を収集することで，従量課金の根拠となる利用情報を収

集する方式である． 利点はエージェントソフトに高度な計測機能を組み込み可能なため監視項目への対応の自由度が高い点である．短所としてはエージェントを従量課金対象サーバに導入するため，対象となるハイパーバイザや OS ごとにエージェントソフトの準備が必要になる点と，エージェントソフトウェア自体が資源を利用するため，クラウド利用者環境に対して，資源使用率上昇，安定性低下，セキュリティ問題などが発生するリスクがある点である．

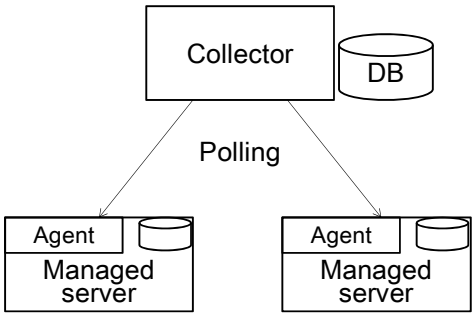

図 8 測定エージェントによる計測

4.2. 監視サーバからの死活監視で計測

監視サーバからネットワーク経由のノード到達性で従量課金対象サーバの状態を監視することで，従量課金の根拠となる利用情報を収集する方式である． 利点は，従量課金対象サーバにエージェントを導入する必要がないため，エージェント導入で生じるクラウド基盤運用上のリスクを回避することができる点である．短所として，この方法で測定できるのはサーバの通電状態であり，サーバ通電時間で課金する方式でしか利用できない．また監視サーバと監視対象間のネットワークが不通になっている期間は利用情報を得られない欠点がある．

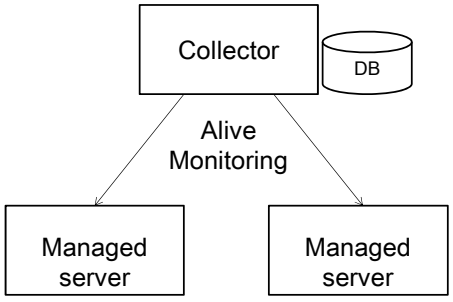

図 9 死活監視による計測

4.3. 各サーバの OS の機能で計測

Linux における Sysstat や， Windows のパフォーマンスモニタのように OS に標準で含まれる測定機能を利用して資源利用状態を監視，ローカルに一時保存しておき，監視サーバで定期的に情報を収集することで，従量課金の根拠となる利用情報を収集する方式である．利点はパフォーマンス測定の監視項目対応の自由度が高い点である．短所としては OS やハイパーバイザーごとにパフォーマンス測定機能が異なるため，監視側は従量課金対象サーバごとに測定方法を実装しなければならない．

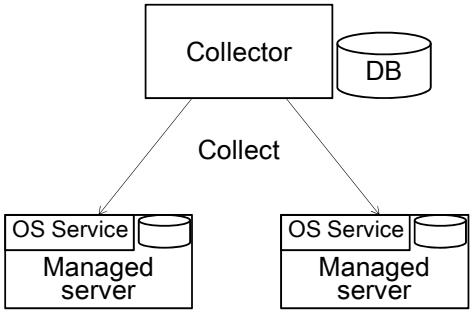

図 10 OS 機能による計測

4.4. 仮想基盤(OpenStack)を介した計測

CPU やメモリ，ディスク，ネットワークやストレージのオブジェクト数，容量など，OpenStack の API を通じて使用量計測を測定する方式である．利点として OpenStack で構築されているクラウド基盤においては，OS や H/W など環境の違いを意識せずに実装できる点にある．例えば Cisco UCS は x86 アーキテクチャ，IBM POWER Systems は POWER アーキテクチャと CPU アーキテクチャは大きく異なるが OpenStack をサポートする仕様が確保できていれば共通の従量課金管理下におくことが可能である．ただし OpenStack 上でのサーバ従量課金については，過去に実装された実績がないため，その実現性が実証されていないのが短所となる．

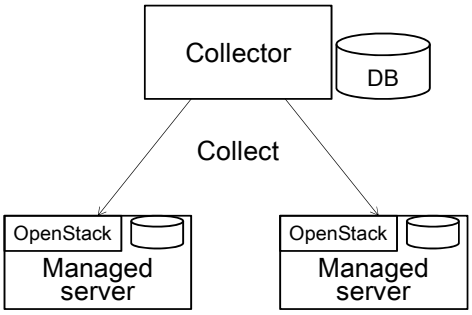

図 11 仮想化基盤機能による計測

4.5. 監視方法のまとめ

ハイブリッド課金オファリングの技術実装において，サーバ利用状況測定とレポート機能について４つの実装方法を検討した．結果として，OpenStack での実装が最も優れているとの結論に至った．ただし実装の実績がないため，概念実証（Proof of Concept)が必要である．

5. OpenStack 環境におけるサーバ利用情報収集(コレクタ）の概念実証

特定の機種や CPU アーキテクチャに依存しない OpenStack 環境の従量課金測定実装としてサーバ利用情報収集機能(コレクタ）の概念実証を行った．OpenStack では，CPU アーキテクチャ，ハイパーバイザに依存しない実装が可能で，Cisco，IBM も含め多くのクラウド事業者や IT ベンダが支持している[17]．OpenStack 自身にも従量課金メータリング機能として「OpenStack Metering」

が実装されているが，同機能はサーバの起動/停止時間の記録は不可能で従量課金に必要な稼働時間測定には現時点では機能不足である. そこで筆者は, OpenStack の枠組みを拡張する形で，ハイパーバイザのログ情報と OpenStack の API から取得できる情報を組み合わせ，ハイパーバイザと仮想マシンの稼働情報を取得する機能をモジュールとして開発し， サーバ利用情報コレクタと同等に OpenStack 環境でサーバ稼働情報の収集可能な実装とした．本論文の概念実証において作成したコレクタのソースコードは本稿の付録として 9 章に添付する．

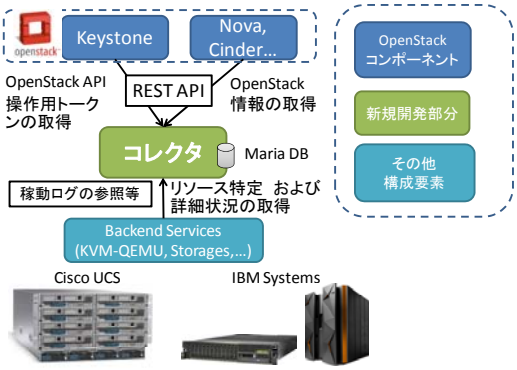

図 12　OpenStack 環境における従量課金測定　概念実証のアーキテクチャ概要

図 12 のようにサーバ利用情報コレクタと OpenStack のインターフェースには REST API を利用している．まず Keystone 認証でサーバ稼働情報取得に必要な権限を取得する．その後，OpenStack からインスタンス(仮想サーバ)稼働情報の取得を行い，MariaDB にログを格納する構造である．稼働情報は，バックエンドサービスから得た情報(KVM インスタンスの名称，ストレージ WWPN 等)と突き合わせて物理リソースを特定する．本概念検証としては，KVM (QEMU) との連携ドライバを作成し，稼働状況を追跡するプログラムを作成した． Nova ドライバから得られるインスタンス情報と KVM のログを，それぞれ定期的に取得・突合し，各インスタンスの起動/停止時間を DB に格納していくもので，アウトプットイメージは表 1 のようになる．

この実装アイデアのポイントは，OpenStack 関連のコンポーネントと，バックエンドサービスとの連携コンポーネントを分離したことで，それにより新規の H/W やハイパーバイザへの対応に必要な改修を最小限に留めることができる．また OpenStack との連携により仮想サーバ単位の起動/停止での課金判断が可能であるため，クラウドでは一般的なマルチテナントに対応可能であることも含め，クラウド事業者の要望に沿える設計となっている．バックエンドサービスとの連携は，ドライバさえ実装可能であればベンダを選ばないが，物理サーバの情報を API 経由で十分に取得できることが望ましい. その観点で, IBM Systems の REST API または Cisco UCS Manager の XML API は実装性や取得可能な情報の充実などからこの方式に向いた機能といえる．

表 1　概念実証用コードで取得した仮想サーバ起動/停止時刻　(テナント UUID は略記)

p	tenant	instname	vmname	valid	flavor	starttime	endtime	closed
1	1	test01	instance-00000005	0	1	2015/8/24 8:07	2015/8/28 8:32	1
2	2	test02	instance-00000006	0	1	2015/8/24 8:12	2015/8/28 8:18	1
3	2	test03	instance-00000007	0	2	2015/8/24 8:17	2015/8/28 8:29	1
4	2	test02	instance-00000006	0	1	2015/8/24 8:21	2015/8/28 8:34	1
5	3	test04	instance-00000008	0	2	2015/8/25 3:11	2015/8/26 15:11	1
6	3	test05	instance-00000009	0	1	2015/8/25 3:18	2015/8/26 15:13	1
7	2	test03	instance-00000007	0	2	2015/8/25 11:25	2015/8/25 12:13	1
8	2	test02	instance-00000006	0	1	2015/8/25 12:18	2015/8/25 14:07	1
9	1	test01	instance-00000005	0	1	2015/8/25 13:07	2015/8/26 15:21	1
10	3	test02	instance-00000006	0	1	2015/8/26 3:04	NULL	0
11	1	test01	instance-00000005	0	2	2015/8/26 7:12	NULL	0

6. ハイブリッド従量課金の効果分析

ハイブリッド従量課金のクラウド事業者における財務上の利点を確認するため，メーカサーバと ODM サーバを比較する投資採算性シミュレーションを行った．シミュレーションではクラウド事業者におけるサーバ 100 台の導入を想定し，以下のパラメータを前提とした．なお採算評価指標は『損益分岐点までの期間』『4 年間の総利益(損失)額』の２点とする．

ODM サーバ購入額:	3600 万円(100 台)
メーカサーバ購入額:	6000 万円(100 台)
サーバ耐用年数:	4 年
撤去・廃棄費用:	1．5 万円/サーバ

事業収益変動は，以下の４パターンと想定し，それぞれにサーバ利用率の変動設定を行った．

段階的成長	事業は初年度から段階的に成長する (1 年目 25%,　2 年目 50%,　3 年目 75%,　4 年目 100%)
垂直立上げ	事業は初年度に急激に成長する (1 年目に 70%,　2 年目 80%,　3 年目 90%,　4 年目 100%)
後期成長	事業は低迷し，最終年に急成長する (1 年目 10%,　2 年目 10%,　3 年目 20%,　4 年目 100%)
事業非成長	事業は低迷し，成長せずに終了する (1 年目 10%,　2 年目 10%,　3 年目 15%,　4 年目 15%)

6.1. ODM サーバ調達時の採算性評価

ODM サーバ調達時のクラウド事業者の事業成長パターン別の採算性評価は図 13 になる．【段

階的成長】と【垂直立ち上げ】では損益分岐点を超え大きな利益を得る，【後期成長】では，ほぼ損益分岐点に達したところで，システムライフが終了し，【事業非成長】では損益分岐点に達せず，多額の損失を計上する結果になった．

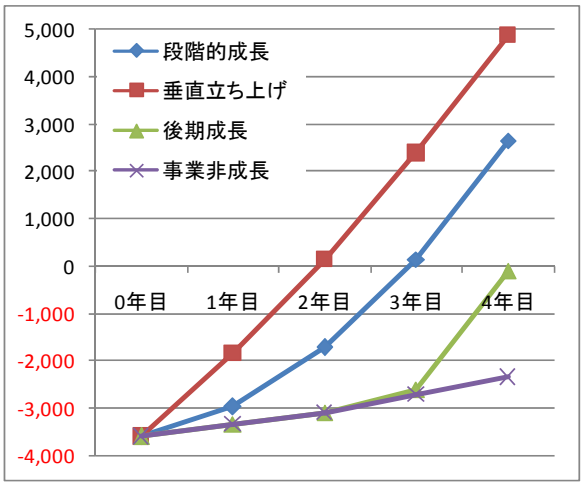

図 13　ODM サーバの損益シミュレーション (万円)

6.2. ハイブリッド従量課金サーバ調達時における採算性評価

ハイブリッド従量課金オファリングによる事業成長パターン別の採算性評価は図 14 になる．特徴は総利益は ODM サーバに劣るが，初期支出が少ないため損益分岐点への到達が ODM サーバより早く，【垂直立ち上げ】は１年目，【段階的成長】は２年目に損益分岐点に達する．また【事業非成長】でもほぼ損益分岐点に達したのは，同成長パターンで大損失になった ODM サーバとは対照的な結果だった．前述のように，莫大な設備投資を前提とするクラウド事業にとって，事業成長が進まなかった場合に損失・被害を最小限にして撤退する出口戦略は不可欠であることを考慮すると，クラウド事業者にとってハイブリッド従量課金は ODM サーバに対して明白な優位点がある．

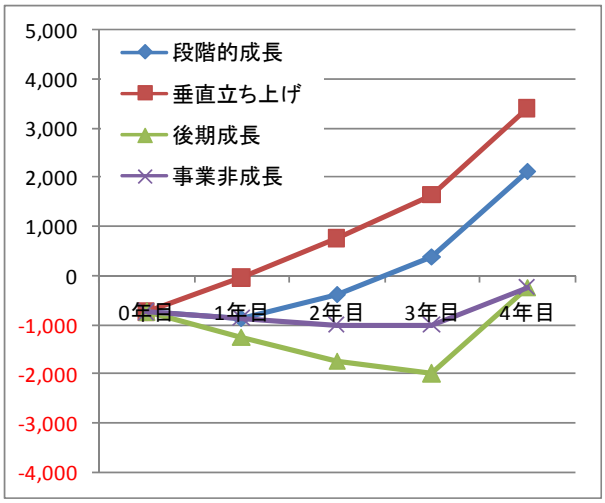

図 14　ハイブリッド従量課金の損益シミュレーション (万円)

6.3. クラウド事業者におけるハイブリッド従量課金の導入効果

これらのコンセプトに基づくハイブリッド従量課金は，実際にクラウド事業者様に採用いただき，事例として発表させていただいている[16]．お客様からの評価で特筆すべき事は，ハイブリッド従量課金がクラウド事業者であるお客様の財務上の利点として，非常に高く評価いただけたことである．この特性は下記のお客様からの評価コメントでも伺え，クラウド事業のお客様におけるハイブリッド従量課金の財務上の高い訴求力有効性が確認できる．

『グループ全体で積み上げると，年間 100 億円以上のコスト削減効果が見込める』[16]

『能力に余裕のあるシステム構成を組むことができ，かつ使用量に応じた支払体系を実現することで，今後は投資効果（ROI)に見合った費用発生にできるため，弊社の事業採算性改善に大いに寄与している』[16]

7. まとめ

近年，サーバ総出荷金額/台数は頭打ちの一方でクラウド事業者のサーバ数は顕著な増加傾向にあるが， 採用サーバは ODM サーバ中心である<1 章>．クラウド事業は高成長，高利益の有望分野だが，サーバに代表される巨額な設備投資は経営上の大きな負担で，メーカサーバより安価な ODM サーバを選択する理由になっている<2 章>．クラウド事業者にサーバを訴求するためのハイブリッド従量課金のコンセプトは，『ハイブリッド・リース』『従量課金』『後払い』の特徴を有し，技術実装は『従量課金対象サーバ』『管理サーバ』『データ管理 PC』の３つのコンポーネントで構成する<3 章>．管理サーバに必要な従量課金測定機能（コレクタ）の技術実装を検討したが OpenStack を介したサーバ利用状況取得が最も適している<4 章>．概念実証として OpenStack から稼働サーバ情報を取得するモジュールを作成し，実現性を確認した<5 章>．投資採算性シミュレーションでは，ハイブリッド従量課金は ODM サーバに対して明確な優位点は投資回収期間が短く，事業非成長時の損失を最小限に抑えられる点が示された．それらの財務上のメリットは実際にサーバ従量課金を導入いただいたお客様からの評価コメントでも裏付けられている<6 章>.

クラウド事業の成熟により，サーバ機器市場は大きな転換期を迎えている．単に顧客が企業/官公庁からクラウド事業者中心に変わるというだけではなく，サーバの価値訴求と競争力を再構築する必要があると考える．それは必ずしも ODM サーバと同じ土俵でサーバの低価格競争に突入することだけを意味するのではない．サーバメーカが，従前のようにクラウド事業者にとってのサーバ機器のサプライヤで有り続けるだけではなく，クラウド事業者の抱えるコーポレート・ファイナンスの課題に共同で取り組む「イノベーション・パートナ」ーになれるかが，サーバメーカのクラウド事業における製品価値再構築の鍵になると考える． 本稿が提言したハイブリッド従量課金はその試みの一つと言えるだろう．

8. 参考文献.

[1] 矢野経済研究所, 『クラウドソーシングサービス市場に関する調査結果』, 30 July 2014

[2] MM総研, 『国内クラウドサービス需要動向』, 2014

[3] 富士キメラ総研, 『データセンタービジネス市場調査総覧』, 2013

[4] Morgan Stanley, *Blue Paper - Cloud Computing Takes Off*, 2011

[5] IDC, *Japan Cloud Vendor Trend of Procurement*, Mar 2015

[6] IDC, *Worldwide and U.S . Server 2013 Vendor Shares*

[7] Dan Gallagher, *Google: Weapon of Choice in the Cloud*, The Wall Street Journal, April 29, 2014

[8] Amazon, *First Quarter Sales 2015*, Apr 2015

[9] Google, *Announces First Quarter 2015 Results*, Apr 2015

[10] Rackspace, *Quarterly report which provides a continuing view*, May 2015

[11] James Hamilton , *AWS re:Invent 2014*, Nov2014

[12] IT pro, WWW ホスティング最大手の米エクソダスが破産申請, Sep 2001

[13] Sonia Lelii, *Nirvanix prepares to close, tells customers to stop using its cloud*, TechTarget, 2013

[14] IDC Japan,サーバベンダーの評価基準, July 2010 http://japan. zdnet. com/article/20416884/

[15] IT pro, 「出そろった国産メーカーのクラウド事業」, July, 2009

[16] 日本 IBM, 導入事例, http://www-03.ibm.com/software/businesscasestudies/jp/ja/jirei?synkey=E923920I03337R71, 2012

[17] OpenStack Fondation, *Companies Supporting The OpenStack Foundation*,2015

9. 付録：OpenStack 環境のサーバ利用情報収集の概念実証用コード

9.1. 実装ソースコード (Python)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
### OpenXOD prototype implementation
### Last updated: Aug. 18, 2015 by Hiroaki Kuramae

import re
import os
import sys
import json
import MySQLdb
from datetime import datetime
from sys import stdout
from httplib import HTTPConnection

### this program should be regularly run (e.g. by for cron)

def setConfig():
""" Define the global configulation variables

Args: None

Returns: A dict mapping keys to the corresponding configulation variables.

Raises: None

"""
    config = {}
    config["username"] = "admin"
    config["tenant"] = "admin"
    config["password"] = "PasswOrd"
    config["keystone_admin_url"] = "kilo-test:35357"

    config["dbuser"] = "openxod"
    config["dbpass"] = "PasswOrd"

    config["libvirt_logpath"] = "/var/log/libvirt/qemu"
    config["logpath"] = "/var/log/openxod/openxod.log"
    return config

##########################################

def qemu_log_parse(logfile):
""" Parse KVM (QEMU) logfile

Args:
Logfile: path to the QEMU logfile

Returns:
    Array that contains tuple with the following format:
    (start_time, end_time)
        start_time: timestamp that VM start in yyyy-MM-DD HH:mm:ss
        end_time: corresponding timestamp that VM destroyed in yyyy-MM-DD HH:mm:ss
    None is returned if some I/O error occurs.

Raises: None

"""

    try:
        f = open(logfile, 'r')
        buff = []
        running = False
        start_str = r"(¥d{4}-¥d¥d-¥d¥d ¥d¥d:¥d¥d:¥d¥d)(.¥d{3}¥+¥d{4})(: starting up)"
        end_str   = r"(¥d{4}-¥d¥d-¥d¥d ¥d¥d:¥d¥d:¥d¥d)(.¥d{3}¥+¥d{4})(: shutting down)"
        for line in f:
            logstr = line.rstrip()
            trial1 = re.match(start_str, logstr)
            trial2 = re.match(end_str, logstr)
            if not running:
                if trial1:
                    buff.append((trial1.group(1), None))
                    running = True
                elif trial2:
                    continue
            else:
```

```
                if trial2:
                    buff[-1] = (buff[-1][0], trial2.group(1))
                    running = False
elif trial1:
buff.append((trial2.group(1),None))
                    continue
        return buff

    except:
        print sys.exc_info()[0]
        return None

def getAdminToken(keystone_admin_url, tenant, username, password):
""" get Admin token JSON instance from Keystone API

Args:
    keystone_admin_url: the admin url to the keystone
    tenant: tenant name (string)
    username: user name (string)
    password: password (string)

Returns:
    JSON instance returned by API

Raises: None

"""

    p = HTTPConnection(keystone_admin_url)
    p.request("POST",
              "/v2.0/tokens",
              "{¥"auth¥":{¥"tenantName¥":¥"%s¥",¥"passwordCredentials¥":{¥"username¥":¥"%s¥",¥"password¥":¥"%s¥"}}}" ¥
              % (tenant,username,password) ,
              {"Content-Type":"application/json"})
    z =  json.load(p.getresponse())
    return z

def getAdminTokenUUID(z):
""" get Admin token UUID

Args:
z: JSON contains the admin token

Returns:
UUID of the access token

Raises: None

"""

    return z["access"]["token"]["id"]

def getNovaServiceUrl(z):
""" get Nova service url

Args:
z: JSON contains the admin token

Returns:
    url string of Nova service

Raises: None

"""

    return [z["endpoints"][0]["publicURL"]
            for z in z["access"]["serviceCatalog"]
            if z["type"]=="compute"][0]

def getServerInfo(novaServiceUrl, token):
""" get JSON contains server detail info

Args:
novaServiceUrl: Nova service URL
    token: admin service token UUID

Returns:
    JSON returned by API

Raises: None

"""
```

```
    u =  re.match(r"^(http|https)://(.*:¥d+)(/.*)$",novaServiceUrl).groups()

    p = HTTPConnection(u[1])
    p.request("GET",u[2]+"/servers/detail?all_tenants=1", "",
              {"Content-Type":"application/json","X-Auth-Token":token})
    z = json.load(p.getresponse())
    p.close()
    return z

def getActiveServerInfo(serverInfo):
""" get running instances list

Args:
serverInfo: JOSN returnced by Nova API

Returns:
Array of tuple corresponding running instances in following format
    (instance name, internal VM name, tenant id, flavor)

Raises: None

"""
    return map(lambda x: (x["name"], x["OS-EXT-SRV-ATTR:instance_name"],
                          x["tenant_id"], x["flavor"]["id"]),
               filter(lambda x:x["status"] == "ACTIVE",
                      serverInfo["servers"]))

def getDBConn(dbuser, dbpass):
""" get DB connection for MySQL/MariaDB

Args:
dbuser: database user
    dbpass: database password

Returns:
MySQL database connection instance

Raises: None

"""

    dbConn = MySQLdb.connect(db="openxod", user=dbuser, passwd=dbpass, charset="utf8")
    return dbConn

def getServerListFromDB(dbConn):
""" get server list from DB

Args:
    dbConn: MySQL database connection instance

Returns:
Array of rows that the instance is not destroyed

Raises: None

"""
    cursor = dbConn.cursor()
    sql = u"select * from data where closed=FALSE"
    cursor.execute(sql)
    return cursor.fetchall()

def logMessage(logpath, msg):
"""Write log message

Args:
    logpath:  path to the log
    msg: message to write

Returns: None

Raises: None

"""

    f = open(logpath, 'a')
    f.write(datetime.now().strftime("[oslogger] %Y.%m.%d %H:%M:%S ")+msg+"¥n")
    f.close()

def insertNewInstEntry(dbConn, tenant, instname, vmname, flavor, starttime):
"""add new instance entry

Args:
```

```
    dbConn: MySQL database connection instance
    tenant: tenant id
    instname: instance name
    vmname: internal VM name
    flavor: flavor
    starttime: instance start time

Returns:  None

Raises: None

"""

    cursor = dbConn.cursor()
    data = (tenant, instname, vmname, flavor)
    sql = u"insert into data (tenant, instname, vmname, valid, flavor, starttime, closed) values(%s, %s, %s, 0,
%s,¥'"+starttime+"¥' , FALSE)"
    cursor.execute(sql, data)

def updateEndedInstEntry(dbConn, p, endtime):
    cursor = dbConn.cursor()
    data = (p)
    sql = u"update data set endtime=¥'"+endtime+"¥', closed=TRUE where p = %s"
    cursor.execute(sql, data)


def main():
    config = setConfig()
    logMessage(config["logpath"], "OpenXOD starts.")

    adminTokenInfo = getAdminToken(config["keystone_admin_url"],
                                   config["tenant"],
                                   config["username"],
                                   config["password"])
    adminTokenUUID = getAdminTokenUUID(adminTokenInfo)
    novaServiceUrl = getNovaServiceUrl(adminTokenInfo)

    serverInfo = getServerInfo(novaServiceUrl, adminTokenUUID)
    activeServerInfo = getActiveServerInfo(serverInfo)

    dbConn = getDBConn(config["dbuser"], config["dbpass"])
    dbEntryList = getServerListFromDB(dbConn)
    _dbEntrySet = set([entry[3] for entry in dbEntryList])

    # All instances that not run before are added to DB
    for vm in filter(lambda x: x[1] not in _dbEntrySet, activeServerInfo):
        vmlogfilename = config["libvirt_logpath"]+"/"+vm[1]+".log"
        starttime_cand = filter(lambda x: x[1]==None,
                                qemu_log_parse(vmlogfilename))
        starttime = starttime_cand[-1][0]
        insertNewInstEntry(dbConn, vm[2], vm[0], vm[1], vm[3], starttime)
        logMessage(config["logpath"], "New instance is created: "+vm[2]+", "+vm[1]+".")

    # All instances destroyed are notified and recorded the end time
    # based on QEMU log.
    for entry in dbEntryList:
        vmlogfilename = config["libvirt_logpath"]+"/"+entry[3]+".log"
        starttime = entry[6].strftime('%Y-%m-%d %H:%M:%S')
        endtime_cand = filter(lambda x: x[0] == starttime,
                              qemu_log_parse(vmlogfilename))
        if endtime_cand[-1][1] != None:
            endtime = endtime_cand[-1][1]
            updateEndedInstEntry(dbConn, entry[0], endtime)
            logMessage(config["logpath"], "Instance destroyed: "+entry[1]+", "+entry[2]+".")


    dbConn.commit()
    logMessage(config["logpath"], "OpenXOD successfully completed.")

if __name__=='__main__':
    main()
```

9.2. MariaDB 向け DDL

```
create database openxod character set utf8;
use openxod;
create table data (p int auto_increment, tenant varchar(255), instname varchar(255), vmname varchar(255), valid int, flavor
int, starttime datetime, endtime datetime, closed boolean, primary key(p), key(instname, starttime));
grant all privileges on openxod.* to 'openxod' identified by 'PasswOrd';
```