社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－HPC－116 (18)
2008／8／6

リソースネームスペース管理サービスの負荷分散手法の提案

中村昌弘[†1] 建部修見[†1]

Open Grid Forum により，名前空間を一元的に管理するための Web サービス，Resource Namespace Service (RNS) が提案された．RNS の仕様では名前空間を複数サーバで管理すること自体は想定されているが，負荷分散を考慮した分散方式に関しては評価が進められていない．また耐障害性に関しては考慮されていない．そこで従来の RNS 仕様との互換性を可能な限り維持したまま，負荷分散および耐障害性を実現する手法を提案する．本稿では，大きく分けて primary-copy 方式と subtree partitioning 方式の２つについて，それぞれ RNS 仕様との親和性と導入する際の問題点について比較，検討する．

Load balancing of Resource Namespace Management Service

Masahiro Nakamura [†1] and Osamu Tatebe[†1]

Open Grid Forum have presented Resource Namespace Service (RNS), which is a web service to manage namespace hierarchy in a unified manner. While the current RNS specification supports distributed servers to maintain a namespace, its capability for load balancing is not evaluated yet. Moreover, it is not intended for fault torerance. We propose distributed load balancing and fault tolerant system into RNS while preserving compatibility to its current specification. We examine expected merits and problems of two methods, which are primary-copy and subtree partitioning, when they are integrated to RNS.

1. はじめに

Open Grid Forum[1] により，Web サービスベースのディレクトリ管理サービス Resource Namespace Service (RNS)[2] が提案された．これはファイルシステムのような仮想的名前空間を管理するための Web サービスである．RNS により，複数の拠点，ストレージに分散したデータを一元的に管理することができる．

現在の RNS は複数のサーバを用いて名前空間を管理することは想定しているものの，負荷分散を考慮した分散方式については評価が進められていない．また耐障害性を考慮したものではないため，耐障害性は確保されていない．

性能面における欠点のひとつに，RNS は単一の仮想ディレクトリを分散させることは想定していないため，単一ディレクトリに対するクライアントからのアクセス増加に対応することができない点である．もうひとつは，広域分散環境において複数の拠点で利用する場合に，それぞれが地理的，ネットワーク的に離れた環境に設置されていた場合，サービスにアクセスするためのレイテンシが増加するという問題がある．耐障害性面では，サービスを提供するサーバが停止すると RNS の機能がすべて失われてしまうという欠点がある．

本稿ではこれらの問題を解決するために，スケーラビリティと耐障害性を持つよう RNS 仕様を複数サーバによる負荷分散および耐障害性確保が可能なものに拡張する．また従来の RNS 仕様と可能な限り互換性を失わないよう配慮する．

2. リソースネームスペースサービスとは

RNS は Web サービスベースのディレクトリ管理サービスである．これはファイルシステムのような仮想的階層名前空間を管理する．ファイルシステムのディレクトリに相当するものを仮想ディレクトリと呼び，ファイルに相当するものをジャンクションと呼ぶ．仮想ディレクトリは名前空間内にのみ存在するため“仮想”と呼ばれる．ジャンクションはリソースへの参照を保存し，ファイルシステムにおけるシンボリックリンクのような働きをする．RNS を用いると，これらを組み合わせてあらゆるリソースを統一的な階層名前空間で管理することができる．

RNS は様々なグリッドアプリケーション等に名前

†1 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

空間を提供し，グリッド環境で参照される仮想化されたデータ，サービスなどのあらゆるリソースを管理することを目的に設計されている．

Web サービスやそれに対応するアプリケーション，その他のサービスリソースが増加し，簡便な手段でこれらにアクセスする必要性がある．これが統一的なリソース名前空間管理サービスが必要とされる理由である．さらにリソースを仮想化することへの評価が高まり，名前から仮想化されたアドレスへの変換機能を提供する RNS のもたらす利益をより大きなものとしている．

RNS は名前空間を階層構造として管理するため，同じく階層構造として管理されているファイルシステムなどとの親和性が高い．従来はリソースを参照するために，個々のリソース固有の表記を取らざるを得なかった．つまり，ファイルシステムと Web サービスなどの異なった種類のリソースを，シームレスな名前空間に対応付けることが不可能であった．RNS では，自身の管理する階層構造の中にそれぞれのサービスへの参照を保存するため，全く性質の異なるリソースでもひとつの名前空間で管理することが可能となる．リソースへの参照を表記するためには endpoint reference (EPR) を用いる．これにより，EPR で表記することのできるあらゆるリソースが単一の階層構造として管理できる．さらに，RNS は自身に保存される EPR の種類について関知しないため，他のサービスによって提供される名前空間や抽象的な識別子を記述することが可能であり，さらに柔軟な参照を可能とする．

RNS は Open Grid Forum によって仕様が策定・公開されている Web サービスであり，仕様に準拠したアプリケーションであればどのようなものでも利用することができる．そのため，複数のアプリケーションで統一的なリソースの管理が可能となる．

3. 分散サーバの必要性

典型的なファイル操作において，名前空間を参照するメタデータ操作はファイル I/O のおよそ半分を占める[9] ため，メタデータ操作のスケーラビリティは重要である．

現在の RNS 実装[3] を用いて，クライアント台数を変化させたときの処理時間は **図 1** の通りである．list はエントリ数によらず一定の性能を示しているが，それ以外の操作はエントリ数が増加するにしたがって性能が悪化し，線形に処理時間が増加している．このことから，多数のクライアントが接続する環境では単体のサーバでは処理しきれないことが想定できる．したがって，数十台のクライアントに対応するためには複数サーバによる負荷分散が必要である．

さらに，広域分散環境で用いることを想定すると，サーバが一カ所にのみ存在するだけでは遠隔地からのアクセスのレイテンシが増加するので，性能が制限される．たとえば日米間には 100ms 以上の RTT が想定され，これがそのままレイテンシの増加につながる．統一された名前空間を複数の拠点から使用したいという需要は多く，サーバを分散させることによってこの制限を緩和する必要性がある．

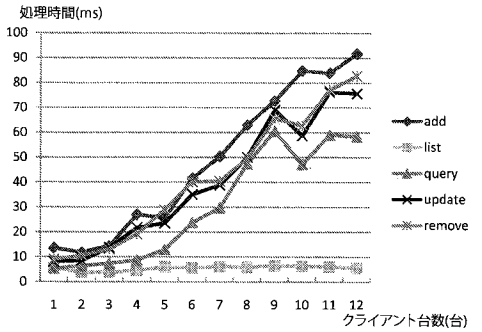

図 1 クライアント台数を変化させた場合のエントリあたり処理時間
クライアント台数を 1 台から 12 台に変化させて測定した．仮想ディレクトリあたりのエントリ数は 256 エントリである．

またこの方式では，サーバやネットワークに障害が発生するとシステム全体が停止するので耐障害性が低いという問題がある．サーバを分散して運用することで，ひとつのサーバに障害が発生しても他のサーバにリクエストを転送し，可運用性を高める必要がある．

4. 負荷分散の方式

負荷分散の方式として，Primary-copy 方式および subtree partitioning 方式を比較，検討する．

4.1 Primary-copy 方式

Primary-copy 方式は、ひとつのサーバがプライマリとなり、他のサーバに名前空間のツリー全体を特定のタイミングでコピーする方式である．仕組みが単純で実装が容易であること，どのサーバも名前空間全体を保持するので，多数の読み込みが発生する場合に性能が向上ことが利点である．反面，書き込みをひとつのサーバでしか基本的に処理できないので，書き込みが多数発生する環境下ではスケーラビリティが確保できないことが欠点である．

Primary-copy 方式はコピーを作成するタイミングによって，さらに次の 4 つの方式に分けられる．

Invalidate based 方式

Invalidate based 方式では，プライマリに更新リクエストが届いた際に，プライマリが複製管理サーバにすべての複製を無効化するよう通信する方式である．複製を管理しているサーバは，クライアントから無効化されたエントリに対するリクエストがあった時点で改めてプライマリに情報を取得する．

次に述べる Update based 方式と比べて，使われな

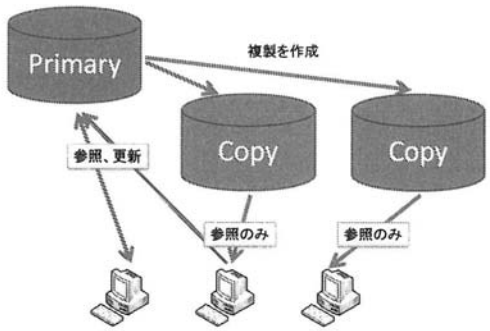

図 2 Primary-copy 方式
名前空間全体を複数のサーバで管理する．プライマリとなったサーバが更新リクエストを受け付け，それ以外のサーバは参照のみを担当する．更新リクエストは適宜複製管理サーバに転送される．

いエントリについては複製管理サーバへ更新された情報が送られないので，プライマリに対してデータを更新した際の負荷が軽減できるのが特徴である．しかしながら，プライマリに対する更新の大部分が複製管理サーバに要求される場合，更新時と参照時の二度にわたって通信が発生してしまうため，逆に負荷が高くなる可能性がある．

Update based 方式

Update based 方式は，プライマリに更新リクエストが届いた際に，すべての複製を更新する方法である．この方式では，複製管理サーバに保存されている情報は常にプライマリと一貫していることが保証される．

複製管理サーバにクライアントが問い合わせた際，最小の手順でクライアントに最新の情報を返すことができるため，書き込みに比べて読み込みが多く発生する環境に適している．その反面，プライマリに対して発行されたすべての書き込みリクエストが複製管理サーバにも転送されるため，書き込みについては単一サーバで構成するよりもパフォーマンスが低下してしまう問題点がある．

Mtime based 方式

Mtime based 方式では，プライマリはディレクトリごとの更新時間を管理し，複製管理サーバはクライアントからのリクエストがあるたびにディレクトリの更新時間をサーバーに確認し，更新時間が新しければディレクトリの複製を更新する．

プライマリに更新があった場合でもプライマリから複製管理サーバに通信する必要が無く，更新リクエストが多い場合は invalidate based 方式や update based 方式よりも適しているといえる．その反面，複製管理サーバにクライアントから読み込みリクエストがあった場合でも必ずプライマリに通信する必要があり，参照リクエストが多い場合のスケーラビリティに欠ける．

TTL based 方式

TTL based 方式は，複製管理サーバのそれぞれの情報について有効期限を定め，有効期限内であればプライマリに確認することなく保持している情報をクライアントに返す手法である．利点は，有効期限内であればプライマリと複製管理サーバ間には一切通信が発生しないので，プライマリの負荷が低くなり，また複製管理サーバがプライマリからネットワーク的に切り離されても一定の動作ができることである．欠点は，有効期限が短すぎるとプライマリと複製管理サーバの間に無用の通信が多数発生してしまうことと，有効期限内であれば古い情報をクライアントに送信するので，一貫性が確保できないことである．

4.2 Subtree partitioning 方式

Subtree partitioning 方式 (**図 3**) は，Dynamic Metadata Management for Petabyte-scale File Systems[5] で提案されている方式で，ツリーの枝ごとにひとつのサーバで扱う方式である．

Primary-copy 方式と比べた利点は，それぞれのサブツリーがひとつのサーバに割り当てられるため一貫性を確保することが容易である点である．Primary-copy 方式ではひとつのディレクトリの内容を複数のサーバが管理するため，プライマリと複製管理サーバの間で一貫性を確保する必要がある．そのため一貫性管理のオーバーヘッドが存在し，スケーラビリティが制限された手法を用いるか，スケーラビリティを確保するためには一貫性をある程度犠牲にする必要があった．反対に subtree partitioning 方式はディレクトリ内の情報を複製しないため，ディレクトリツリーが大きければサーバの台数を増やすことが可能で，負荷を分散できる．欠点として，ひとつのディレクトリに負荷が集中した場合には全く負荷を分散できないことと，ひとつのディレクトリを複数の拠点で使用した場合にはどちらかの拠点からのレイテンシが増加することが挙げられる．これらの欠点を解決するためには，他の手法を取り入れる必要がある．

静的 subtree partitioning 方式

静的 subtree partitioning 方式は，あらかじめどのサブツリーをどのサーバに割り当てるかを決めておく手法である．たとえば，NFS でマウントポイントごとに別の NFS サーバを割り当てることに相当する．

利点はサブツリーの割り当てが頻繁に変更されることがないため，サブツリー割り当てのオーバーヘッドが少ないことである．サーバとクライアントで共通のサブツリー割り当てマップを保持しておくことによって，直接サーバにリクエストを送信することができる．またサーバも動的に負荷を分散させる必要がないため，処理コストを負担しなくて良い．仕組みが単純なので，実装が用意であることも利点としてあげられる．その反面欠点として，あらかじめ均等になるようにサブツリーをそれぞれのサーバに割り当てることが大変困難

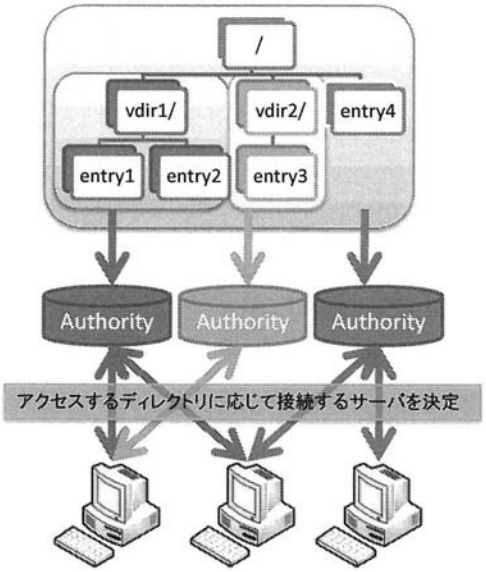

図 3 Subtree partitioning 方式
名前空間のサブツリーが，それぞれのサーバに割り当てられる

であるということが挙げられる．またサービスの提供期間を通して負荷が一定である場合には負荷を均等に分散させることができるが，それぞれの負荷が変化する場合には本方式では十分に対応することができない．

動的 subtree partitioning 方式

動的 subtree partitioning[4)5)] 方式は，どのサブツリーをどのサーバに割り当てるかを実行中に変更する方式である．利点は，実際の利用形態に応じた負荷分散となるため，各サーバの負荷を均等に保つことができることである．また事前に予測の必要がないため，設定が静的 subtree partitioning 方式よりも簡単にできる．

欠点は静的 subtree partitioning 方式とは反対に，サブツリーの割り当てがサービスの提供中に変更されるため，クライアントに何らかの手段で目的のサブツリーが割り当てられているサーバを知らせる必要がある．またサブツリーの割り当てを負荷に応じて変更することもオーバーヘッドとなる．さらに仕組みが静的 subtree partitioning よりも複雑であるので，実装も複雑となることが問題として挙げられる．

4.3 ハッシュ方式

分散ファイルシステムで広く用いられている手法として，ハッシュ方式が挙げられる[10)11)]．ハッシュ方式は，それぞれのエントリの EPR やエントリ名などの属性を基準にして，ハッシュ関数を用いて割り当てるサーバを定める方式である．利点として，ハッシュ関数をあらかじめ定めておくとすべてのサーバとクライアントが計算のみでサーバ割り当てを決めることができる点が挙げられる．動的 subtree partitioning 方式はサーバに問い合わせなければ，あるサブツリーがどのサーバに割り当てているかを知ることができないので，ハッシュ方式はより通信量を削減することができる．

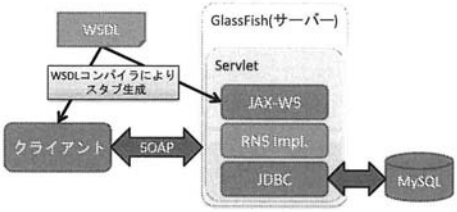

図 4 RNS 実装詳細

欠点は，仮想ディレクトリやジャンクション単位でサーバ割り当てが変わってしまうので，たとえばディレクトリ内のエントリ一覧を取得するといった操作を行うときに複数のサーバに問い合わせる必要があることである．また，ハッシュ関数の偏りによってはひとつのサーバに負荷が集中する可能性がある．

ハッシュ方式においては，サーバの割り当て単位を仮想ディレクトリ単位やエントリ単位と変化させることができる．後者では，ひとつのディレクトリを複数のサーバに割り当てることができるが，反面，一回のサーバへの操作でディレクトリ内のエントリをすべて取得することができないことが欠点として挙げられる．また複数拠点で使用する場合は，それぞれのサーバが同一拠点にあるとは限らず，分散させたことによりレイテンシの増加による性能低下を引き起こす可能性がある．

ハッシュ方式は単体で利用するだけではなく，subtree partitioning 方式と組み合わせ，一部のディレクトリだけをハッシュによって分散させる方式も考えられる．

5. 本研究の実装

本研究では，先行研究 3) で提案した RNS 実装を基本とする．実装の詳細を**図 4** に示す．本実装では web サービス向けのフレームワークとして JAX-WS[7)] を用い，名前空間を管理するためのデータベースとして MySQL Community Server 5.1[8)] を用いる．サービスを動作させるためのアプリケーションサーバには Glassfish v2[6)] を用いる．

5.1 Primary-copy 方式の実装

Primary-copy 方式を実装する際に，4 つの手法を比較評価する必要があるため，基本設計は共通化して実装を簡略化する．

データベース

Web サービス内部でエントリを保存するために，

表 1 データベーススキーマ

型	列名	説明
BIGINT	**id**	エントリ ID
VARCHAR(255)	name	エントリ名
BIGINT	parent	親仮想ディレクトリ
TINYINT(1)	isValid	エントリが有効かどうか
DATETIME	TTL	エントリの TTL
DATETIME	mTime	エントリの最終更新時刻
TINYINT(1)	isDir	仮想ディレクトリなら 1

MySQL Community Server 5.1 を利用した．スキーマを **表 1** に示す．RNS 仕様書によると，更新操作においては名前空間をアトミックに変更することが要求されており，データベースを更新する操作もアトミックに行う必要があるため，テーブル形式として InnoDB を採用した．

id は 64bit 整数型のエントリ ID で，エントリが追加されるごとに異なる正の番号が割り当てられる．また，ルートディレクトリの ID は 1 で固定である．name にはエントリ名が utf-8 として保存される．長さの最大値は 255 文字である．parent は親ディレクトリの ID (ルートディレクトリの場合は負の数) が保存される．isDir はエントリが仮想ディレクトリかどうかを示すビットである．isValid は invalidate based 方式でエントリの有効性を示すためのビットである．TTL は TTL based 方式においてエントリの有効時間を保持するためのフィールドである．mTime はエントリの最終更新時間を保持するためのフィールドである．Mtime based 方式においてはディレクトリが更新されているかどうかを調べるために用いる．

クライアントがサーバに読み込みリクエストを発行する際は，次の手順によって処理する．

(1) クライアントはサーバの IP アドレスを問い合わせる．DNS ではひとつのホスト名に複数の IP アドレスを割り当てることができるので，単一の EPR に複数の複製管理サーバを割り当てることできる．
(2) クライアントは要求されたサーバに接続する．
(3) 接続されたサーバは名前空間全体を保持しているので，読み込みリクエストであればすべて 従来の RNS と同じように処理することができる．

書き込みリクエストを発行する場合は，接続する手順は読み込みと同じであるが，リクエストの処理方法が従来と異なる．

(1) 読み込みと同様にサーバに接続する．
(2) 書き込みリクエストがプライマリではない複製管理サーバに発行された場合，RNSCrossServiceFault が発生する．
(3) クライアントは RNSCrossServiceFault に含まれているプライマリサーバの情報を使用して，プライマリサーバに接続し，更新する．

複製管理サーバが直接プライマリサーバに接続しない理由は，書き込みが大量に実行される場合にはクライアントがプライマリサーバに直接アクセスした方が性能上有利なためである．従来のクライアントは書き込みは処理できないが，読み込みはすべて処理できる．

```
<xsd:complexType name="
    RNSCrossServiceFaultType">
  <xsd:complexContent>
    <xsd:extension base="
        rns:RNSFaultType">
      <xsd:sequence>
        <xsd:element name="
            PrimaryServer" type="
            xsd:string" minOccurs=
            "0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

図 5 RNSCrossServiceFault の定義

また，クライアントにプライマリサーバの所在を通知するため RNSCrossServiceFault を**図 5.1** のように拡張する．

RNSFaultType は WS-BaseFaults[12] を拡張しており，“任意の要素をその中に含むことができる”と定義されているため，PrimaryServer 要素を追加しても従来の仕様との互換性は失われない．

複製管理に対応しているクライアントは，RNSCrossServiceFault に含まれる PrimaryServer のアドレスを参照し，再度更新リクエストを発行する．PrimaryServer はプライマリ RNS の URI である．

Invalidate, update based 方式ではプライマリサーバがクライアントから更新リクエストを受け取ると，それぞれ登録されている複製管理サーバに通知を行う．一貫性を保証するために，すべての複製管理サーバに通知が完了してからクライアントに操作結果を返す．逆に，Mtime, TTL based 方式ではプライマリサーバ自身の情報を更新した時点でクライアントに操作結果を返す．

5.2 Subtree partitioning 方式

Subtree partitioning 方式を実現する上で重要なことは，たとえ仮想ディレクトリの担当サーバが変更されたとしても，過去の RNS で使われていた EPR が引き続き使用可能であることである．本方式においてサーバの割り当てが変更された場合エントリの EPR が変更されるが，本論文では WS-Naming 仕様[13] を用いてクライアントに変更された EPR を通知する．

静的 subtree partitioning 方式

静的 subtree partitioning 方式では，それぞれの RNS サーバが従来と同様に動作するように設計する．

仮想ディレクトリを指すエントリが，従来では同じサーバ内のアドレスのみを参照していたのに対して，違うサーバを指すように構成するだけで実現できる．ある仮想ディレクトリがそれぞれのRNSサーバでルートディレクトリに相当する場合，従来は自分自身への参照をクライアントに返していたが，上位のディレクトリを担当するサーバへのリファレンスを返すよう変更する．

静的subtree partitioning方式は，実現が容易で導入することによるオーバーヘッドがほとんど存在しないことが特徴である．またクライアントは従来のものがそのまま利用できることも，大きな利点として挙げられる．

動的 subtree partitioning 方式

動的subtree partitioning方式では，それぞれのエントリを担当するサーバが随時変更されるため，静的subtree partitioning方式のように，エントリのEPRが直接割り当てサーバを表す方式では，名前空間の一貫性を確保できない．

動的subtree partitioning方式を実現するためには，定期的に各サブツリーを担当するサーバの負荷を比較し，負荷が大きいサーバは自分のサブツリーの一部を他のサーバに委譲する必要がある．委譲するときにはクライアントからのリクエストを一時的に受け付けないようにした状態で，委譲するエントリを移譲先のサーバに転送する．

通常のファイルシステムでは，パスがオブジェクトの階層を保持しているので，パスの一部を保持しておくだけでそれより下の階層がどこに委譲されているのか情報というを得ることができる．一方RNSでは，EPRは階層構造を保持するとは限らないため，EPRからサーバへの割り当てを得ることは困難である．たとえば，通常のファイルシステムでは `/usr/local` は `/usr` の下の階層であるということが自明であるが，RNSでは `http://example.net/RNS?id=12` が `http://example.net/RNS?id=123` と階層構造の中でどのような関係になっているのを把握することができない．

そこでEPRが階層構造を表さないという前提のもとで動的subtree partitioning方式を適用させるために，WS-Naming仕様を用いてクライアントに変更後のEPRを通知する．クライアントは必要に応じて割り当て結果をキャッシュすることによって性能を向上させることができる．

6. おわりに

現在のRNSが単一サーバによって割り当てられた名前空間を管理していることによる，スケーラビリティおよび耐障害性の問題を解決するために，RNSを複数のサーバで分散管理する手法を提案した．その際，現在のクライアントとの互換性をできるだけ維持するように配慮した拡張を行った．

今後の課題は，これらの手法を実装し実際の性能評価を行うことである．ハッシュ方式については具体的な実装方針を考察し，他の方式と比較検討および組み合わせた手法について評価する予定である．また互換性の面において問題がないか，従来の使用に基づくクライアントとの相互運用性の検証も行う必要がある．さらに，従来のRNS仕様との互換性を配慮することなく拡張を行った際，互換性を失うに値するだけの性能向上が果たせる手法があるかどうかについて検討する予定である．

謝辞 本研究の一部は，情報爆発時代に向けた新しいIT基盤技術の研究，文部科学省科学研究費補助金「特定領域研究」(課題番号19024009)による．

参考文献

1) Open Grid Forum. `http://www.ogf.org/`
2) M. Pereira, O. Tatebe, L. Luan and T. Anderson. "Resource Namespace Specification". GFD.101, OGF, 2007
3) 中村 昌弘, 建部 修見. リソースネームスペース管理サービスの実装. 情報処理学会全国大会 2008
4) Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long. Ceph: A Scalable, High-Performance Distributed File System. In Proceedings of the 7th conference on USENIX
5) S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic metadata management for petabyte-scale filesystems. In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04). ACM, Nov. 2004
6) GlassFish - Open Source Application Server. `https://glassfish.dev.java.net/`
7) jax-ws: JAX-WS Reference Implementation. `https://jax-ws.dev.java.net/`
8) MySQL. `http://www.mysql.com`
9) D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In Proceedings of the 2000 USENIX Annual Technical Conference, pages 41-54, San Diego, CA, June 2000. USENIX Association
10) P. J. Braam. The Lustre storage architecture, 2002
11) P. Schwan. Lustre: Building a file system for 1000-node clusters. In Proceedings of the 2003 Linux Symposium, July 2003
12) Lily Liu and Sam Meder, Web Services Base Faults 1.2 (WS-BaseFaults), OASIS Standard, April 1 2006
13) A. Grimshaw, D. Snelling. WS-Naming Specification. GFD.109, OGF, July 3 2007