Vol.2016-OS-138 No.7
2016/8/8

ASURA：スケールアウト型分散ストレージ向け データ分散アルゴリズム

石川健一郎[†1]

概要：スケールアウト型分散ストレージで必要になる大量のデータとデータを記憶したサーバの対応の管理はテーブルでは難しく，適切に設計されたアルゴリズムによる管理が求められる．本論文ではデータを冗長記憶するスケールアウト型分散ストレージのために設計したデータ分散アルゴリズム ASURA を提案する．このアルゴリズムは次の特徴を持つ．1）データが冗長化されている場合においてもサーバ構成変更時に最小限のデータ移動のみ行う．2）計算時間は 0.4μ秒未満であり，オーダーは O(1)になる．3）データが十分あるときサーバ間のデータ分散のばらつきは 0.5%程度になる．4）サーバ容量に合わせてデータを分散できる．評価の結果，データを冗長記憶するスケールアウト型分散ストレージでは，類似アルゴリズムである Consistent Hashing，Random Slicing や Weighted Rendezvous Hashing などと比べて ASURA は優れた特性がある事を示した．

キーワード：データ分散アルゴリズム，スケールアウト型ストレージ

ASURA : Data Distribution Algorithm for Scale-out Distributed Storage

KEN-ICHIRO ISHIKAWA[†1]

1. はじめに

コンピュータが取り扱う情報量が爆発的に増加するにつれ, ストレージシステムの容量も増大している. そのため，今日のストレージシステムが要求される容量を 1 台や少数台のストレージサーバで実現する事は難しくなっている．これを解決するため，多数のストレージサーバを 1 台のストレージシステムとして取り扱う分散ストレージ技術が求められている．

分散ストレージの問題の一つとして，データとデータを記憶したサーバの対応の管理が挙げられる．テーブルを用いる手法もあるが，ここではスケーラビリティに優れる手法としてデータ分散アルゴリズムを用いる手法を考える．

データ分散アルゴリズムを用いる手法では，データ ID とサーバ構成を入力としてデータ分散アルゴリズムを実行し，データを記憶するサーバを得る．データ分散アルゴリズムの中でも Consistent Hashing [4]がデファクトスタンダードとして用いられている．Consistent Hashing は次のような特徴を持っており，様々なスケールアウト型分散ストレージ[10] [11]で実用化されている．1）データ ID とサーバ構成からデータを記憶するサーバを一意に決定できる．2）データを冗長化した状態でほとんどの場合で最小限のデータ移動でサーバ構成変更に対応できる．3）計算時間が短い．4）サーバ間のデータ数のばらつきが小さい．5）サーバの容量の違いにおおまかに対応できる．だが，最小限のデータ移動でサーバ構成に対応できない可能性があるため，それに対応するための処理が必要になる，細かいサーバの容量の違いに対応できない，サーバ間のデータ数のばらつきが小さいながらもあるため，あるサーバが一杯になったとき他のサーバには容量に余裕がある状態になり，ストレージの容量を有効活用できないなどの難点がある．Random Slicing [5]や Weighted Rendezvous Hashing [6]などこれらの難点を解消したアルゴリズムも発表されているが，前者はデータを冗長化したときに最小限のデータ移動でサーバ構成を変更できない，後者は計算コストが高いなどの問題がある．本論文では挙げた全ての難点を解消するデータ分散アルゴリズムとして ASURA（Advanced Scalable and Uniform storage by Random number Algorithm）を提案する．

本論文では，ASURA のアルゴリズムを解説し，ベンチマークを取って同種のアルゴリズムである Consistent Hashing，Random Slicing や Weighted Rendezvous Hashing と比較する．その結果，データを冗長記憶するスケールアウト型分散ストレージ向けデータ分散アルゴリズムとして ASURA は優れた特性を持っていることを示す.

- ☆ Consistent Hashing, Random Slicing 及び ASURA の計算時間は 1000 サーバある時 0.4μs 未満である. Weighted Rendezvous Hashing の計算時間は 1000 サーバある時 36.7μs 程度である.
- ☆ データが十分あるとき，Random Slicing，Weighted Rendezvous Hashing 及び ASURA のサーバ間のデータ数のばらつきは 0.5%程度である．Consistent Hashing は 1.7%から 36%のばらつきがある.
- ☆ データをサーバ間で冗長化しているとき，Weighted Rendezvous Hashing と ASURA ではサーバ構成変更時に移動するデータ量は最小限である．Consistent

†1 日本電気株式会社　NEC Corporation

Hashing と Random Slicing ではサーバ構成変更時に移動するデータ量は最小限にならない場合がある．

論文の構成は次のとおりである．2 章で ASURA のアルゴリズムを説明し，3 章で定性的な評価を行う．そして，4 章で定量的な評価を行い，5 章で議論をする．次に 6 章で関連研究について述べ，7 章でまとめる．

2. ASURA のアルゴリズム

この節では ASURA の具体的なアルゴリズムについて記述する．

まず，2.1 節で ASURA の基本的な考え方を示す．2.2 節で ASURA の基本的なアルゴリズムについて述べる．そして, 2.3 節で ASURA の疑似乱数の値域が限られている問題の解決方法である ASURA 乱数について述べる．最後に，2.4 節で ASURA 乱数の生成方法について述べる．

2.1 ASURA の基本的な考え方

ASURA の基本的な考え方は次の通りである．

ASURA では疑似乱数生成器にデータ ID をシードとして与え，サーバを割り当てた値域に入るまで疑似乱数の生成を繰り返し，疑似乱数列が入った値域に割り当てたサーバにデータを記憶する．このような方式をとることにより次のような性質を実現する．まず，疑似乱数であるため，データ ID が同一であれば疑似乱数列も同一になり，サーバ構成が同じであればデータを記憶するサーバも同一になる．また，詳細は 2.2 節で議論するが，最小限のデータ移動でサーバ構成を変更できる．さらに，疑似乱数は高速に生成することが可能なため計算時間は短く，疑似乱数の偏りは極めて小さいためサーバ間のデータ数のばらつきは小さい．加えて，サーバの容量に比例した長さの値域をサーバに割り当てることにより，サーバの容量に合わせた確率でサーバにデータを記憶できる．これらの考え方は SPOCA [1]と同一である．

ASURA ではさらに，サーバが増え，サーバを割り当てた値域が疑似乱数の値域を超える場合，もしくは，サーバが減り，サーバが割り当てられた値域が疑似乱数の値域と比べて非常に小さくなり疑似乱数生成回数が増大する場合に対応するため，疑似乱数の値域を拡大縮小することを可能にする．これにより，サーバが増えた場合にも対応できるスケーラビリティとサーバが減った場合に疑似乱数を振る回数を減らす効率性を両立する．

2.2 節から 2.4 節では以上のような特徴を持った ASURA の具体的なアルゴリズムについて述べる．

2.2 ASURA の基本アルゴリズム

本節では ASURA のアルゴリズムについて述べ，ASURA の次の性質を示す．

・サーバの容量に比例した数のデータを分散する
・サーバ構成変更時に最小限のデータのみ移動する

本節で説明するアルゴリズムは SPOCA と似ている．

ASURA のアルゴリズムは２つのステップに分けられる．

STEP 1.　数線上でサーバとセグメントを対応付ける
STEP 2.　データを記憶するサーバを決定する

STEP 1 は初期化ステップであり，ストレージを開始するとき，サーバ構成を変更するときに実行する．このステップではサーバを数線上のセグメントと次のルールに基づき対応付ける．

１．　サーバは一つのセグメントもしくは複数セグメントに対応付ける．セグメントの長さの合計はサーバの容量に比例する．
２．　サーバの容量の変更のために必要な場合を除き，ストレージに参加済みのサーバとすでに対応付けたセグメントの組み合わせは変えない．
３．　セグメントは数線上の整数のポイントから始まる．セグメントが始まるポイントの数をセグメント番号とする．
４．　セグメントの長さは 1.0 未満とする．

ルール 3 とルール 4 は必須のルールではないが，これらのルールによりセグメントのサーチが簡単になる．

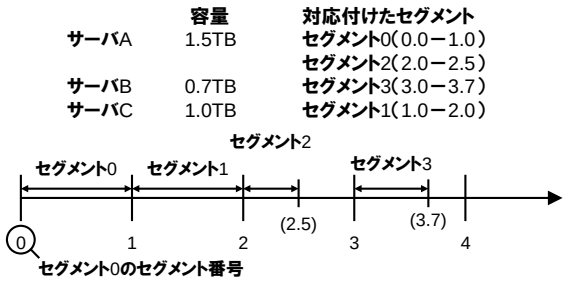


図 1：サーバとセグメントの対応

図 1 はサーバを数線上のセグメントと対応付けた例である．X-Y は X 以上 Y 未満を表す．

STEP 2 のアルゴリズムは次の通りである．

１．　データ ID を使って疑似乱数生成器を初期化する
２．　疑似乱数を生成する
３．　疑似乱数があるセグメントの値域に入っている時，そのセグメントに対応付けられているサーバにデータを記憶する
４．　疑似乱数がセグメントの値域に入っていない時，２に戻る

STEP 2 ではセグメントに含まれるまで疑似乱数を生成する．理論的には STEP 2 が終わらない可能性があるが，十分一様な疑似乱数生成器を使い，極端なセグメント配置を行わなければ，実際上は処理が終わらない事はない．十分一様な疑似乱数生成器の例としてはメルセンヌツイスター [2]，xorshift [3]などが挙げられる．

例えば，図 1 の時，次のような疑似乱数列を生成するデータ ID を持つデータ A とデータ B があるとする．

データ A の乱数列：4.2, 1.1, …

データ B の乱数列：3.3, 0.6, …

このとき，データ A はセグメント 1 と対応しているサーバ C が記憶し，データ B はセグメント 3 と対応しているサーバ B が記憶する．

このアルゴリズムにより次のような性質を実現できる．

1． データはサーバの容量にほぼ比例した数，各サーバに分散する
2． サーバを追加したとき，追加したサーバが記憶する事になるデータのみサーバ間を移動する．このとき，サーバの容量にほぼ比例した数だけのデータを記憶する
3． サーバを削除したとき，削除したサーバに記憶していたデータのみサーバ間を移動する．このとき，サーバの容量にほぼ比例した数だけのデータを記憶する

始めの性質は明らかである．疑似乱数がセグメントに含まれる確率はセグメントの長さに比例し，セグメントの長さはサーバの容量に比例する．

2 番目の性質は ASURA のアルゴリズムが疑似乱数をセグメントに含まれるまで繰り返し生成する事に依存する．

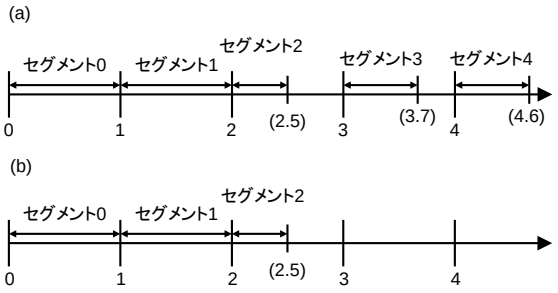


図 2：サーバの追加と削除

図 1 の状態から 0.6TB の容量を持つサーバ D を追加し図 2 (a) のようにセグメント 4 を追加したとする．このとき，セグメント 4 の値域に疑似乱数が含まれる前にセグメント 0 から 3 の値域に疑似乱数が含まれるデータはサーバ D を追加した影響を受けない．だが，セグメント 0 から 3 の値域に含まれる前にセグメント 4 の値域に疑似乱数が含まれるデータは記憶するサーバがサーバ D になる．つまり，記憶するサーバがサーバ D になるデータ以外は記憶するサーバは変わらない．また，各サーバに記憶するデータの数はセグメントの長さ，つまり各サーバの容量に比例する．データ A とデータ B の例ではデータ A は記憶するサーバがサーバ C からサーバ D に移動するが，データ B は影響を受けない．

データを冗長化している場合でも同様の議論ができる．ASURA ではデータを冗長化するため，必要な台数サーバを選択するまで疑似乱数を生成し続ける．サーバを追加したとき，データを記憶するサーバは次の場合に場合分けできる．

（追加したサーバが記憶するサーバとして選ばれていないとき） ： 変わらないか追加したサーバに変化する

（1 つ前にサーバを選択するデータの記憶するサーバが変化したとき） ： 1 つ前にサーバを選択するデータを記憶していたサーバに変化する

つまり，冗長化したデータのうち 1 データを記憶するサーバが追加したサーバに移動するか，冗長化した全てのデータを記憶するサーバが変わらないかの二択である．

3 番目の性質も ASURA のアルゴリズムが疑似乱数をセグメントに含まれるまで繰り返し生成する事に依存する．

図 1 の状態からサーバ B を削除するため図 2 (b) のようにセグメント 3 を削除したとする．このとき，疑似乱数列の疑似乱数を前から評価するとき，セグメント 3 の値域に乱数が含まれる前にセグメント 0 から 2 の値域に疑似乱数が含まれるデータはサーバ B を削除した影響を受けない．だが，疑似乱数列の疑似乱数がセグメント 0 から 2 の値域に含まれる前にセグメント 3 の値域に疑似乱数が含まれるデータは記憶するサーバがサーバ B から異なるサーバに変更になる．つまり，記憶するサーバがサーバ B だったデータ以外は記憶するサーバは変わらない．また，各サーバに記憶するデータの数はセグメントの長さ，つまり各サーバの容量に比例する．データ A とデータ B の例ではデータ B は記憶するサーバがサーバ B からサーバ A に変化するが，データ A は影響を受けない．

データを冗長化している場合でも同様の議論ができる．サーバを削除したとき，データを記憶するサーバは次の場合に場合分けできる．

（記憶するサーバとして選んでいたサーバが選択できるとき） ： 変わらない

（記憶するサーバとして選んでいたサーバが選択できないとき） ： 1 つ後にサーバを選択するデータを記憶していたサーバに移動する

つまり，削除したサーバを記憶するサーバとして選んでいたデータよりも後にサーバを選択するデータは，データを記憶するサーバが 1 つ後にサーバを選択するデータを記憶していたサーバに変更になり，最後にサーバを選択していたデータのみ今までデータを記憶していなかったサーバに記憶される．

つまり，冗長化したデータのうち 1 データを記憶するサーバが今までデータを記憶していなかったサーバに変化するか，冗長化した全てのデータを記憶するサーバが変わらないかの二択である．

これらの性質から ASURA はサーバ構成変更時に最小限のデータ移動でサーバの容量にほぼ比例した数各サーバにデータを分散できる．

また，これらの性質は疑似乱数が次の条件を満たすとき実現できる．この条件を ASURA 条件と呼ぶ．

1． 疑似乱数は値域内で一様である

２． シード（データ ID）が同じとき，値域内の疑似乱数は同じ数列になる

だが，ASURA の基本アルゴリズムは疑似乱数の値域が固定されているため，スケーラビリティと効率性の両立ができない．疑似乱数生成器の値域を狭く取った場合，少数のセグメントしか疑似乱数の値域に置くことができないため，スケーラビリティを達成できない．また，疑似乱数生成器の値域を広く取った場合，疑似乱数の値域に対してセグメントがカバーする地域が狭くなるため，疑似乱数の発生回数が多くなり効率性を達成できない．そこで次節では ASURA の基本アルゴリズムのスケーラビリティと効率の両立の問題を解決する ASURA 乱数について説明する．

2.3 ASURA 乱数

この節では ASURA で用いる ASURA 乱数について記述する．ASURA 乱数は特別な方法を用いて疑似乱数の値域の拡大縮小を可能にする．ASURA 乱数により ASURA のスケーラビリティと効率の両立の問題の解決が可能になる．

まず，ASURA 乱数における値域の拡大について議論する．ASURA 乱数において値域を拡大するためには ASURA 条件を保つ必要がある．

ASURA 条件を満たした上で値域を拡大するために，元の値域を持つ疑似乱数列に拡大した値域の疑似乱数を疑似乱数が一様に分散するように挿入した疑似乱数列を考える．

例えば，図 1 のようにサーバとセグメントが対応付けられている場合，次のような 0.0-4.0 の値域を持つ疑似乱数生成器が使用可能である．このとき，疑似乱数列が次のようになるとする．

疑似乱数列：2.7, 3.8, 1.1…

このとき，データはサーバ C に記憶する．次に，データ ID を元にした値で初期化した 4.0-8.0 の値域を持つ疑似乱数生成器を使って，疑似乱数列を生成する．そして，疑似乱数が一様になり，生成した疑似乱数列を既存の値域の疑似乱数は値も順番も変化しないように挿入すると，この疑似乱数列は例えば次のようになる．

乱数列：5.4, 2.7, 4.3, 3.8, 1.1…（下線部が挿入した乱数）

この疑似乱数列を用いた場合，疑似乱数列の値域が変化した後でもデータを記憶するサーバはサーバ C から変化しない．

一般に，上記の方法で乱数列の値域を変更しても元の乱数列の値域にあるセグメントに対応付けたサーバが記憶するデータは記憶するサーバは変化しない．

また，上記の方法で疑似乱数列の値域を変更した疑似乱数列は ASURA 条件を満たす．つまり，この方法を用いる事により，ASURA で用いる疑似乱数生成器の値域を拡大する事ができる．

疑似乱数の値域の縮小も同様に議論できる．ASURA ではセグメントが存在しない値域の疑似乱数を疑似乱数列から削除しても，データを記憶するサーバは影響を受けない．つまり，セグメントが存在しない値域の疑似乱数を削除し，疑似乱数の値域を縮小する事が可能である．また，このとき，ASURA 条件を満たす事は明らかである．

つまり，この方法により ASURA で用いる ASURA 乱数の値域の拡大縮小が可能になる．セグメントが増えたとき，スケーラビリティを達成するため，ASURA 乱数は値域を拡大する事が可能である．また，セグメントが減ったとき，効率を上げるため，ASURA 乱数は値域を縮小する事が可能である．つまり，ASURA は ASURA 乱数によりスケーラビリティと効率性の両立が可能になる．次の節では ASURA 乱数の生成方法について説明する．

2.4 ASURA 乱数生成アルゴリズム

ASURA 乱数を生成するためには次のような疑似乱数生成器群を用意する．

１． 各疑似乱数生成器は ASURA 乱数を生成するためのシードが同じであれば同じ乱数列を生成し，異なる場合は偶然の一致を除いて異なる乱数列を生成する

２． より値域が広い疑似乱数生成器の値域はより値域が狭い疑似乱数生成器の値域を包含する

３． 少なくとも一番値域が広い疑似乱数生成器は生成したい ASURA 乱数の値域を包含する

ASURA 乱数は次のように生成する．

１． 生成したい ASURA 乱数の値域を包含する最も値域が狭い疑似乱数生成器を選ぶ

２． 疑似乱数を生成する

３． 次に値域が狭い疑似乱数生成器があり，疑似乱数が次に値域が狭い疑似乱数生成器の値域内であれば，その疑似乱数生成器を選び，2 へ戻る．

４． 次に値域が狭い疑似乱数生成器がない，もしくは疑似乱数が次に値域が狭い疑似乱数生成器の値域外の場合はその疑似乱数を ASURA 乱数とする

例として，一番値域が狭い疑似乱数生成器の値域を 0.0 以上 8.0 未満とし，以下，値域の最大値が 2 倍になっていくとして，同じシード A を用い，0.0 以上 8.0 未満の ASURA 乱数，0.0 以上 16.0 未満の ASURA 乱数，0.0 以上 32.0 未満の ASURA 乱数を生成するとする．

まず，0.0 以上 8.0 未満の ASURA 乱数を生成する．まず，使用する疑似乱数生成器を選択する．ここでは値域が 0.0 以上 8.0 未満の疑似乱数生成器 0 のみを用いる．次に，シード A をシードとして疑似乱数生成器で疑似乱数を生成し，その疑似乱数を疑似乱数生成器 0 のシード A_0 とする．そして，疑似乱数生成器 0 を用いて疑似乱数を生成する．疑似乱数生成器 0 よりも値域の狭い疑似乱数生成器は存在しないのでその疑似乱数が ASURA 乱数となる．生成した ASURA 乱数の例を次に挙げる．

ASURA 乱数：4.9, 1.1, 2.7…

次に同じシードを使って 0.0 以上 16.0 未満の ASURA 乱

数を生成する. まず, 使用する疑似乱数生成器を選択する. ここでは疑似乱数生成器 0 と値域が 0.0 以上 16.0 未満の疑似乱数生成器 1 を用いる. 次に，シード A をシードとして疑似乱数生成器で疑似乱数を生成し，その乱数を各疑似乱数生成器のシード A_0, シード A_1 とする. そして, まず, 疑似乱数生成器 1 を用いて疑似乱数を生成する. そして, もし, その疑似乱数が疑似乱数生成器 0 の値域外であれば, その疑似乱数を ASURA 乱数とし，その疑似乱数が疑似乱数生成器 0 の値域内であれば，疑似乱数生成器 0 を用いて疑似乱数を生成し，その乱数を ASURA 乱数とする．生成した ASURA 乱数の例を次に挙げる.

ASURA 乱数：4.9, 12.3, 10.4, 1.1, 2.7, 8.2…

（下線を引いた乱数が疑似乱数生成器 1 によって生成した疑似乱数）

ここで，その ASURA 乱数のうち 0.0 以上 8.0 未満の ASURA 乱数は，値域が 0.0 以上 8.0 未満の時の ASURA 乱数と値も並びも全く同じになる.

さらに同じシードを使って 0.0 以上 32.0 未満の ASURA 乱数を生成する．まず，使用する疑似乱数生成器を選択する．ここでは疑似乱数生成器 0，疑似乱数生成器 1 と値域が 0.0 以上 32.0 未満の疑似乱数生成器 2 を用いる．次に，シード A をシードとして疑似乱数生成器で疑似乱数を生成し，その疑似乱数を各疑似乱数生成器のシード A_0～シード A_2 とする．そして，同様に疑似乱数生成器 2 を使って疑似乱数を生成し，その疑似乱数が疑似乱数生成器 1 の値域内であれば疑似乱数生成器 1 を使って疑似乱数を生成し, その疑似乱数生成器 1 が生成した疑似乱数が疑似乱数生成器 0 の値域内であれば疑似乱数生成器 0 を使って疑似乱数を生成し, 最後に生成した疑似乱数を ASURA 乱数とする.

生成した ASURA 乱数の例を次に挙げる.

ASURA 乱数 : 4.9, 21.4, 12.3, 10.4, 1.1, 31.9, 17.7, 2.7, 8.2…

（下線を引いた疑似乱数が疑似乱数生成器 1 によって生成した疑似乱数，二重下線を引いた疑似乱数が疑似乱数生成器 2 によって生成した疑似乱数）

ここで，この ASURA 乱数のうち 0.0 以上 16.0 未満の ASURA 乱数は，値域が 0.0 以上 16.0 未満の時の ASURA 乱数と値も並びも全く同じになる．また，疑似乱数生成器 0 が生成した疑似乱数が ASURA 乱数となる確率は 1/4 であり，疑似乱数生成器 0 が生成した疑似乱数が ASURA 乱数として採用される値域も全体の 1/4 である．同様に疑似乱数生成器 1 が生成した疑似乱数が ASURA 乱数となる確率は 1/4 であり，疑似乱数生成器 1 が生成した疑似乱数が ASURA 乱数として採用される値域も全体の 1/4 である. さらに，疑似乱数生成器 2 が生成した疑似乱数が ASURA 乱数となる確率は 1/2 であり，疑似乱数生成器 2 が生成した疑似乱数が ASURA 乱数として採用される値域も全体の 1/2 である．つまり，ASURA 乱数は一様である.

以上の例から明らかなように疑似乱数生成器を増やす事により，ASURA 乱数は ASURA 性質を満たしながら値域を拡大する事ができる.

値域の縮小も同様に議論する事ができる事は明らかである.

以上より，ASURA は 1 章で示した，1）データ ID とサーバ構成からデータを記憶するサーバを一意に決定できる. 2）データを冗長化した状態で最小限のデータ移動でサーバ構成変更に対応できる．3）サーバの容量の違いに対応できる．といった条件を満たす.

3. ASURA の定性的な評価

この章では Consistent Hashing，Random Slicing，Weighted Rendezvous Hashing，ASURA の定性的な評価を行う. Consistent Hashing はスケールアウト型分散ストレージのデータ分散アルゴリズムのデファクトスタンダードである. Random Slicing と Weighted Rendezvous Hashing はデータ ID とサーバ構成からデータを記憶するサーバを一意に決定でき，最小限のデータ移動でサーバ構成変更に対応でき，サーバの容量の違いに対応できる既存のアルゴリズムである. ここではデータ分散アルゴリズムを次の項目で評価する.

・実行時間，分散の一様さ，表現できる容量の精度，データの冗長化に対する対応

3 章と 4 章ではサーバの数を N，Consistent Hashing で用いるバーチャルノードの数を V とする．また，適宜, Consistent Hashing は CH, バーチャルノードは VN, Random Slicing は RS, Weighted Rendezvous Hashing は WRH と略す.

3.1 実行時間

アルゴリズムの初期設定はサーバ構成が変更されない限り行われないため，実行時間は問題にならない．だが，データを記憶するサーバを決定する処理はデータへのアクセスが行われるたびに行われるため，実行時間は十分に短い必要がある．そのため，本節ではデータを記憶するサーバを決定する時間について議論する.

CH はソートされた VN からデータを置いた領域を持つ VN を発見する必要がある．これはバイナリサーチなどで行うことができ，オーダーは O(log(NV))である．著者らは O(1)のアルゴリズムも利用できると主張している [7].

RS では線分を各サーバの容量に合わせて区切るが, 区切りの数は各サーバの容量に依存し，簡単な式で表すことはできない．そこで区切りの数を NX と置き，バイナリサーチなどでデータを置いた領域を持つサーバを探索するとオーダーは O(log(NX))となる．また，CH と同様に O(1)のアルゴリズムも利用可能である.

WRH では各サーバについて疑似乱数を元に計算した値を求め，最も値が大きいサーバを探す．これは O(N)のオーダーで実行することが可能である.

ASURA では ASURA 乱数を生成し，ASURA 乱数が指す領域を持つサーバにデータを記憶する．ASURA 乱数の生

成は例えば$\sum_{n=1}^{\left\lceil log_2\left\lceil \frac{N}{16} \right\rceil \right\rceil+1} \frac{1}{2^{n-1}}$回の疑似乱数生成により行うことができ，これは定数に収束する．また，ASURA 乱数がセグメントの値域に含まれるか否かはサーバの数に依存しない．つまり，ASURA の実行時間のオーダーは O(1)である．

3.2 分散の一様さ

本節ではどの程度の精度で設定した容量に比例した数のデータを分散できるか評価する．

CH ではサーバが持つハッシュリング上の領域の長さにばらつきが生じ，ハッシュリングに置くデータの位置にもばらつきが生じる．つまり，二重でばらつきが生じる．

RS ではハッシュラインに置くデータの位置にばらつきが生じる．つまり，一重でばらつきが生じる．

WRH では各サーバに割り当てられる疑似乱数にばらつきが生じる．つまり，一重でばらつきが生じる．

ASURA は ASURA 乱数にばらつきが生じる．つまり，一重でばらつきが生じる．

3.3 表現できる容量の精度

本節ではサーバの容量をアルゴリズム上でどの程度の精度で表現できるか評価する．

CH では VN の数を用いてサーバの容量の違いを表現するが，VN の数は 3 桁から 4 桁程度であり，高い精度では表現できない．

RS では領域の長さでサーバの容量の違いを表現するため，高い精度でサーバの容量の違いを表現することができる．

WRH ではアルゴリズムに与える値によってサーバの容量の違いを表現し，これは高い精度でサーバの容量の違いを表現することができる．

ASURA ではサーバに割り当てるセグメントの長さでサーバの容量の違いを表現することができ，これにより高い精度でサーバの容量の違いを表現することができる．

3.4 データの冗長化対応

いずれのアルゴリズムもデータを冗長化するためにデータを記憶する複数のサーバを選択することができる. だが, このとき，データを冗長化しない場合と比較し，サーバ構成変更時の挙動が異なってくる．

CH ではほぼ全ての場合でデータを冗長化している場合でもサーバ構成変更時のデータ移動は最小限である．VN のハッシュリング上の位置が衝突した場合，最小限にならない場合があるが非常にまれである．

RS ではデータを冗長化するとサーバ構成変更時のデータ移動が最小限にならない場合がある．

WRH ではデータを冗長化した場合でもサーバ構成変更時のデータ移動は最小限である．

ASURA でも同様にデータを冗長化した場合でもサーバ構成変更時のデータ移動は最小限である．

3.5 まとめ

アルゴリズムの定性的な評価をまとめると表 1 のようになる．ASURA が定性的な評価では優れた特性を示す.

4. ASURA の定量的な評価

この章では CH，RS，WRH，ASURA の定量的な評価を行う．ここではデータ分散アルゴリズムを次の項目で評価する．

・実行時間，分散の一様さ，データの冗長化対応

4.1 評価環境

定量的な評価で用いた実験環境は次の通りである．

CPU：Intel Xeon X5550 2.67GHz　メモリ：24GB
OS：CentOS 6.3
コンパイラ最適化オプション：-O6 –march=native
疑似乱数生成器：dSFMT（メルセンヌツイスター）
疑似乱数生成器オプション：-DSFMT_MEXP=521

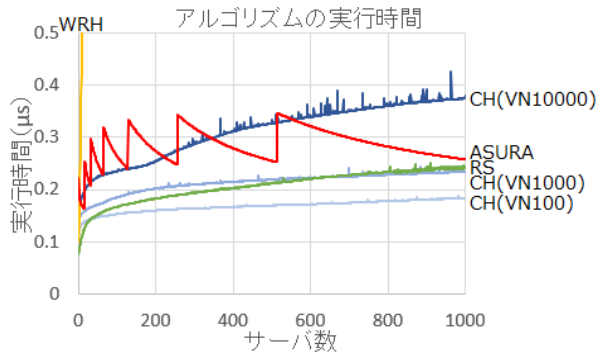
4.2 実行時間

この節ではアルゴリズムの実行時間を評価する．データ分散アルゴリズムはアクセスするサーバを決定するたびに実行されるため，十分短い実行時間である必要がある．また，ネットワークやストレージなどでの処理時間と比較し十分短い実行時間であれば細かい違いは問題にならない.

アルゴリズムの実行時間を測定するために，データ ID とサーバ構成を入力としデータを記憶するサーバの番号を出力する関数を 1,000,000 回実行し，実行時間を測定するベンチマークを 100 回実行し, 実行時間が短いもの 40 回と長いもの 40 回の結果を捨て，残り 20 回の平均を取った. サーバ数 1 台から 1000 台まで測定した. 全てのサーバの容量は一定としている. ASURA ではセグメントの長さは 1.0, セグメント番号は 0 から連続しているとし, 初期の ASURA 乱数の値域は 0.0 以上 16.0 未満として，ASURA 乱数の値域は 2 倍に拡大していくとした. CH と RS はバイナリサーチを用いてノードを検索した．CH のバーチャルノードの

表 1：定性的な評価

	実行時間	分散の一様さ	表現できる容量の精度	データの冗長化対応
CH	O(1) or O(log(NV))	二重のばらつき	低精度	ほぼ良い
RS	O(1) or O(log(NX))	一重のばらつき	高精度	良くない
WRH	O(N)	一重のばらつき	高精度	良い
ASURA	O(1)	一重のばらつき	高精度	良い



グラフ 1：アルゴリズムの実行時間

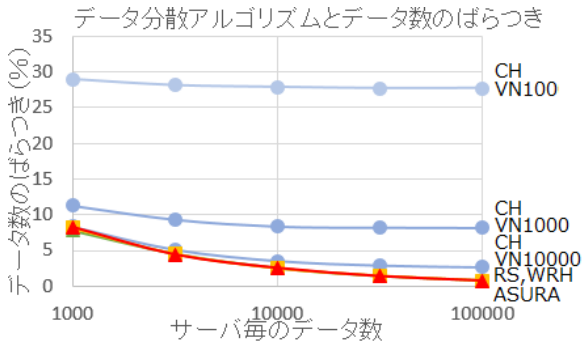
数は 100, 1000, 10000 と設定した．

結果はグラフ 1 のようになった．他のアルゴリズムの実行時間の違いを示すため，WRH の結果のほとんどはグラフ外になっている．CH，RS，ASURA は短い実行時間となった．WRH はサーバ数が少ないときのみ短い実行時間となった．CH 及び RS は O(1)のアルゴリズムも利用可能であるが，より簡単な log オーダーのアルゴリズムで十分実用的である．ASURA はサーバ数が増え，ASURA 乱数の値域とセグメントが存在する領域が一致するにつれ実行時間が短くなり，セグメントが存在する領域が ASURA 乱数の値域を越え，ASURA 乱数の値域を拡大すると急激に実行時間が長くなる．CH と RS では実行時間が安定してスパイクするサーバ数が存在した．

4.3 分散の一様さ

この節では各サーバが記憶するデータの数の一様さを評価する．

アルゴリズムの分散の一様さを測定するために，データ ID とサーバ構成を入力としデータを記憶するサーバの番号を出力する関数を実行し，各サーバの番号の出力回数を集計し，誤差が最大のサーバの誤差率（ばらつき）を評価した．サーバ数は 10 台，100 台，1000 台とし，データ数はサーバ数に 1000, 3162, 10000, 31622, 100000 をかけた数とした（注：$10^{0.5} = 0.31622$）．CH の VN 数は 100, 1000, 10000 とした．評価は 100 回行い，平均を取った．



グラフ 2：分散の一様さ

サーバ台数 100 台の時の結果をグラフ 2 に表す．十分なデータ数があるとき，RS，WRH，ASURA では誤差は 0.5%から 1.0%程度になった．だが，CH は十分なデータ数がある場合でも VN が 100 の時，15%から 36%，VN が 1000 の時，5%から 11%，VN が 10000 の時，1.7%から 3.5%と大きなものになった．

4.4 データの冗長化対応

この節ではデータを冗長記憶しているときにサーバ構成変更した場合，サーバが変更になるデータの数を評価する．データの冗長化対応を評価するためにサーバが 8 台から 9 台に増加するとき，もしくはサーバが 9 台から 8 台に減少するとき，冗長記憶しているデータを記憶するサーバが変化する台数を評価した．10,000,000 データを 3 重冗長で記憶しているとし，10 回シミュレーションし，結果を加算した．結果は表 2 のようになった．A はサーバ追加を R はサーバ削除を意味する．各列は 3 重冗長しているデータのうち，サーバ構成変更時にサーバを移動したデータ数を示す. RS は複数台データを記憶するサーバが変更になる可能性があるが，他のアルゴリズムはデータを記憶するサーバに変更がないか，1 台のみ移動するにとどまった．CH は理論的には複数台データを記憶するサーバが変更になる可能性があるが，今回の評価ではこのような現象は発生しなかった．似たアルゴリズムである CH と RS で違いがある理由は，CH はデータを冗長記憶するサーバを決定する際，データを記憶するサーバのハッシュリング上の隣にあるサーバを選択するのに対し, RS では疑似乱数をさらに生成しデータを記憶するサーバを選択するためである. RS ではデータを記憶するサーバのハッシュライン上の隣にあるサーバにデータを記憶する場合，データがサーバに偏って記憶されるため，CH と同じ手法は取れない．

	0	1	2	3
CH-A	66553833	33446167	0	0
CH-R	66554519	33445481	0	0
RS-A	66667393	31397312	1901370	33925
RS-R	66680268	31387070	1899271	33391
WRH-A	66666890	33333110	0	0
WRH-R	66669256	33330744	0	0
ASURA-A	66661333	33338667	0	0
ASURA-R	66669190	33330810	0	0

表 2：冗長記憶したデータのサーバ構成変更時の移動データ数

4.5 まとめ

定量的な評価ではアルゴリズムの単体の実行時間では CH，RS，ASURA が優れた特性を示した．サーバ間のデータ数の最大誤差では RS，WRH，ASURA が優れた特性を示した．データを冗長記憶した場合の特性では CH，WRH，ASURA が優れた特性を示した．結論として，全ての項目

で ASURA が優れた特性を示した

5. 議論

5.1 分散の一様さの意味

各サーバが記憶するデータの数が一様ではない場合，あるサーバの容量が一杯になったとき，他のサーバは容量に余裕がある状態になる．そのため，ストレージの容量を有効活用できない．各サーバが記憶するデータの数が一様な場合，あるサーバの容量が一杯になったとき，他のサーバの容量もほぼ一杯になるため，ストレージの容量を有効活用できる．

5.2 データ毎のデータサイズ及びデータアクセス頻度の違い

ASURA ではサーバの容量にほぼ比例した数のデータを各サーバが記憶する．だが，データはサイズ及びアクセス頻度に違いがある．この問題は大数の法則によって軽減される．大数の法則により各サーバが十分な数のデータを記憶することによりサイズの違い及びアクセス頻度の違いは平均化される．大数の法則は他のデータ分散アルゴリズム [4] [5] [6]でも前提となっている法則である．

5.3 サーバ数が増えた場合のサーバを移動するデータの判定

サーバ数が増えるに従いデータ数も増大するため，サーバ構成が変更になった際にサーバを移動するか否か判定するデータの数も増えていく．これは各サーバで並列にデータがサーバを移動するか否か判定できることにより解決する．データが増えると共にサーバも増えるため，サーバ構成変更時に移動するデータの判定に必要な時間はデータ数に比例しては増えない．

5.4 サーバ構成変更時に冗長記憶しているデータが複数移動することの意味

サーバ構成変更時に冗長記憶しているデータがサーバを移動しないまたは 1 データのみ移動すると断定できる場合，プログラムの構造が簡単になる．だが，複数もしくは全て移動する可能性がある場合，これを考慮する必要があるため，プログラムの構造が複雑になり，バグの入る余地が広がり，ストレージが不安定になる可能性が高くなる．

6. 関連研究

データをアルゴリズムによって分散配置するシステムは RAID を用いたシステムもあるが，ここでは分散ストレージ向けデータ分散アルゴリズムについて述べる．

最大サーバ数と効率性のトレードオフが存在せず，サーバ構成変更時に最小限のデータ移動のみ必要とするアルゴリズムとしては Consistent Hashing [4]，Highest Random Weight [8]などが初期に開発された．ハッシュリングやハッシュライン上にサーバの領域を設定し，領域に置かれたデータをサーバが記憶する Consistent Hashing 系のアルゴリズムとしては Random Slicing [5]などがある．データ ID と各サーバの ID をハッシュしたものをアルゴリズムで処理し，最も大きな値になったサーバに記憶する Highest Random Weight 系のアルゴリズムとしては CRUSH [9]の Straw Buckets や Weighted Rendezvous Hashing [6]などがある．また，ライン上のセグメントに含まれるまで疑似乱数を振る方式としては SPOCA [1]が初期の研究であり，ASURA はこれの発展系になる．

7. おわりに

スケールアウト型分散ストレージ向けデータ分散アルゴリズム ASURA を提案した．ASURA はデータを冗長記憶している場合でもサーバ構成変更時に最小限のデータのみ移動し，スケーラビリティと効率性を両立している．同様なアルゴリズムと比較した結果，データの冗長記憶に対応したスケールアウト型分散ストレージ向けデータ分散アルゴリズムとして最も優れた特性を持つことがわかった．

参考文献

[1] Chawla, Ashish, et al. "Semantics of caching with spoca: a stateless, proportional, optimally-consistent addressing algorithm." Proceedings of the 2011 USENIX conference on USENIX annual technical conference. USENIX Association, 2011.
[2] M. Matsumoto and T. Nishimura, “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator,” ACM Trans. Model. Comput. Simul. 8, 1, January 1998, pp. 3-30.
[3] G. Marsaglia, “Xorshift rngs,” Journal of Statistical Software, 8.14, 2003, pp. 1-6.
[4] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and Daniel Lewin, “Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web,” In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, STOC 1997, New York, NY, USA, pp. 654-663.
[5] Miranda, Alberto, et al. "Random Slicing: Efficient and Scalable Data Placement for Large-Scale Storage Systems." ACM Transactions on Storage (TOS) 10.3 (2014): 9.
[6] http://www.snia.org/sites/default/files/SDC15_presentations/dist_sys/Jason_Resch_New_Consistent_Hashings_Rev.pdf
[7] Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." ACM SIGCOMM Computer Communication Review 31.4 (2001): 149-160.
[8] Thaler, David G., and Chinya V. Ravishankar. "Using name-based mappings to increase hit rates." IEEE/ACM Transactions on Networking (TON) 6.1 (1998): 1-14.
[9] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, “CRUSH: controlled, scalable, decentralized placement of replicated data,” In Proceedings of the 2006 ACM/IEEE conference on Supercomputing, SC 2006, New York, NY, USA, Article 122 .
[10] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon's highly available key-value store.,” In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, SOSP 2007, New York, NY, USA, 2007, pp. 205-220.
[11] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” SIGOPS Oper. Syst. Rev. 44, 2, April 2010, pp. 35-40.