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

Sequential Checking：サーバ構成変更時にデータ移動しない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：増大するデータを記憶するため安価で大容量なテープデバイスや光学デバイスをスケールアウト型分散ストレージとして使用することが考えられる．しかしながら，これらのデバイスはデータの書き換えが困難もしくは不可能である．このため，スケールアウト型分散ストレージで標準的に用いられるサーバ構成変更時にサーバ間でサーバに記憶したデータを移動するデータ分散アルゴリズムを適用することは難しい．そこで，本論文ではストレージの構成変更時にデータの移動が困難又は不可能なデバイスを用いた分散ストレージに適用可能なデータ分散アルゴリズム Sequential Checking を提案する．Sequential Checking はサーバ構成変更時にデータを移動することなく，サーバの容量に合わせてデータを書き込むことができる．さらに，データを書き込むサーバを一意に決定することができ，データ読み込み時にデータを記憶したサーバを必ず含む多くの場合で少数のサーバを判定可能であり，常に同じデータ ID に結びつけられた最新のデータにアクセス可能である．シミュレーションの結果，Sequential Checking の基本的な特徴を確認することができ，設定した条件下で 256 サーバあるとき平均約 1.98 サーバにアクセスすることによりデータを読み込むことができる事を確認した．この結果より，Sequential Checking により従来不可能だったテープデバイスや光学デバイスを用いたスケールアウト型分散ストレージが実現できると考える．

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，テープストレージ，光学ストレージ

Sequential Checking : A Reallocation Free Data Distribution Algorithm for Scale-out Storage

KEN-ICHIRO ISHIKAWA[†1]

1. はじめに

大容量のストレージの設計において，データを安価に取り扱うために容量を増やす際にコストがリニアに近い形で増えるスケールアウト型分散ストレージを利用する事が考えられる．スケールアウト型分散ストレージを用いることにより，安価なデバイスを用いたサーバをスケールアウト型分散ストレージに追加していくことにより大容量のストレージを低コストで実現することができる．

スケールアウト型分散ストレージのコストをさらに下げ，容量を増やすために，安価で大容量なテープデバイスや光学デバイスを使うことが考えられる．だが，これらを用いたスケールアウト型分散ストレージを実現するのは難しい．スケールアウト型分散ストレージでは大量のデータを取り扱うためにデータとデータを配置したストレージを管理する際データ分散アルゴリズムが使われることが多い [1] [2]. 既存のデータ分散アルゴリズムを用いる場合，ストレージを構成するサーバの構成を変えるたびに既にサーバが記憶しているデータの一部を他のサーバに移すために書き換える．だが，テープデバイスや光学デバイスはデータの書き換えが不可能な事が多く，可能な場合でも書き換えに伴う断片化のコストが高い．そのため，既存の技術ではテープデバイスや光学デバイスを用いてスケールアウト型分散ストレージを実現することは難しい．

この問題を解決するためには，サーバ構成変更時にデータを移動しない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を開発する必要がある．この際，実用のためには，データ分散アルゴリズムはサーバの容量に合わせてデータを書き込むことができ，データの読み込み，書き込みとも現実的なコストで行え，同じデータ ID を持った新しいデータを見分けることができる必要がある．

そこでストレージの構成変更時にデータの移動が困難又は不可能なデバイスを用いた分散ストレージに適用可能なデータ分散アルゴリズム Sequential Checking を提案する．

Sequential Checking は次のような特徴を持つ．

1．サーバ追加時及びサーバの空き容量変更時に既に書き込まれたデータを移動しない
　原理的に書き込まれたデータを移動せずに既にあるサーバの削除や空き容量を超えた容量の削除はできないため，書き込まれたデータを移動せずに可能なサーバ構成変更はサーバの追加及びサーバの空き容量変更のみである
2．各サーバにデータを書き込む回数はほぼ各サーバの空き容量に比例した回数になり，これによりストレージの容量が一杯になるとき各サーバにデータを書き込む回数はほぼ各サーバの容量に比例する
3．一意にデータ書き込み先を決定できる
4．多くの場合，すべてのサーバにアクセスすることなく

†1　日本電気株式会社　NEC Corporation

少数のサーバにアクセスすることにより確実に書き込まれたデータを読み込むことができる

5．常にデータ ID に結びつけられた最新のデータにアクセスすることができる

Sequential Checking を用いることにより，サーバ構成変更時にすでに書き込まれたデータの移動が困難もしくは不可能なデバイスであってもデータ分散アルゴリズムを用いたスケールアウト型分散ストレージが可能になる．

この論文の構成は次のようになっている．2 章では Sequential Checking のアルゴリズムについて記述する．3 章で評価を行い，4 章で考えられる議論について記述する．5 章で関連研究について記載し，6 章でまとめる．

2. Sequential Checking のアルゴリズム

この章では Sequential Checking の具体的なアルゴリズムについて議論する．まず 2.1 節で基本的な考え方について説明し，2.2 節で使用するパラメータの計算方法，2.3 節でデータ書き込み時のアルゴリズム, 2.4 節でデータ読み込み時のアルゴリズムについて説明し, 2.5 節でデータを冗長記憶する方法について説明する．

2.1 Sequential Checking の基本的な考え方

本節では Sequential Checking の基本的な考え方について，アルゴリズムの概略を説明しながら述べる．

サーバ構成変更時にデータを移動しないためには，サーバ構成が変わった場合でも，あるデータ ID を持ったデータを書き込む可能性のあったサーバをデータ読み込み時に読み込みの対象として選び続ける必要がある．また，このとき，サーバを書き込み先として選ぶ確率はサーバの空き容量に比例しなければならない．また，同じデータ ID を持った最新のデータを読み込むことができる必要がある．Sequential Checking ではこれらの特性を次のようにして達成する．

まず，各サーバにはストレージに追加した順にサーバ番号を 0 から順に割り当て，主に読み込みのために SRP（Server Read Probability）と書き込みのために SWP（Server Write Probability）と呼ぶパラメータを設定する．SWP には該サーバの空き容量を該サーバと該サーバよりも先に追加したサーバの空き容量で割った値を設定する．そして, SRP は該サーバの SWP の最大値を設定する．0 番サーバの SRP と SWP は必ず 1.0 になる．例えば，全てのサーバの空き容量が等しい場合，サーバ番号 0 から 4 のサーバの SWP はそれぞれ, 1.000, 0.500, 0.333, 0.250, 0.200（小数点以下 3 桁）となり，このとき SRP も SWP と同じ値になる．

データを書き込む際には，各サーバにサーバの容量に比例した回数データを書き込むようにサーバを選択する．これは次のように実現する．

疑似乱数生成器を使ってデータ ID とサーバ番号から一意に決まる値をシードとして 0.0 以上 1.0 未満の疑似乱数を生成する．そして，サーバ番号が大きな順に SWP と比較し，SWP の方が大きいとき，該サーバを書き込み先サーバとして設定する．

このとき，SWP は該サーバの空き容量を該サーバと該サーバより先に追加したサーバの空き容量で割った値であるため，該サーバに書き込まれる確率は該サーバの空き容量に比例する．つまり，各サーバに書き込まれる回数はサーバの空き容量にほぼ比例する．これはストレージが一杯になるとき，各サーバの容量に比例した回数データを書き込むことを示す．

例えば，前記の例の場合，サーバ番号 4 のサーバは SWP が 0.200 なので 20%の確率でデータが書き込まれる．そして，サーバ番号 3 のサーバはサーバ番号 4 のサーバが選ばれなかった確率，すなわち 80%の確率でデータ書き込み先か判定が行われ，SWP が 0.250 なので 20%の確率でデータが書き込まれる．以下，全てのサーバが同じ 20%の確率でデータが書き込まれる．

データを読み込む際にはデータを書き込んだサーバから確実にデータを読み込むようにサーバを選択する．

データを書き込む際と同様に疑似乱数生成器を使ってデータ ID とサーバ番号から一意に決まる値をシードとして 0.0 以上 1.0 未満の疑似乱数を生成する．このとき，生成する疑似乱数はデータ ID とサーバ番号が同じであればデータを書き込むときと同じ疑似乱数になる．そして，サーバ番号が大きな順に SRP と比較し，SRP の方が大きいとき，該サーバからデータを読み込む．各サーバに割り当てた疑似乱数は書き込むときも読み込むときも同じであり，SRP は必ず SWP 以上であるため，データを書き込んだサーバは必ずデータの読み込み先として選ぶことができる．

また，読み込んだデータが最新のデータであることを保証するため，書き込んだデータよりも先に読まれる同じデータ ID を持つデータを消去する必要がある．消去はデータの管理領域にフラグを立てるなどの方法で行われる．消去が必要になることによる容量への影響は後述の評価で評価する．

このアルゴリズムにより，Sequential Checking は 1 章で挙げた性質を達成する. 以下にアルゴリズムの詳細を書く．

2.2 SRP 及び SWP の決定

SRP 及び SWP は次のように決定する．

1． 各サーバに 0 からサーバ番号を振る．一度振ったサーバ番号は変更しない

2． X 番サーバの空き容量を V_X とするとき，Y 番サーバの SWP である SWP_Y を次のように設定する

SWP_Y = V_Y/SUM(V_0:V_Y)

3． Y 番サーバの SRP である SRP_Y（初期値 0.0）は次のように設定する

if SRP_Y < SWP_Y then SRP_Y = SWP_Y

計算式から明らかなように 0.0≦SRP, SWP≦1.0 である．

また，SRP_0＝1.0，SWP_0＝1.0 である．

2.3 データ書き込み時のアルゴリズム

データを書き込むサーバ及びデータを削除するサーバは次のアルゴリズムで決定する．

STEP1. データ ID と各サーバのサーバ番号を元に 0.0 以上 1.0 未満の疑似乱数を生成し，各サーバに割り当てる

疑似乱数の生成方法はデータ ID と各サーバのサーバ番号を関数で演算した結果を疑似乱数のシードとするなどがある．また，データ ID をシードとして生成した疑似乱数をサーバ番号順にサーバに割り当てる方法もある．可能であれば STEP.2 で使用する疑似乱数のみ生成すれば良い．

疑似乱数生成方法としてはメルセンヌツイスター[3] [4]，xorshift [5]などが考えられる．

STEP2. サーバ番号が大きな順に SWP とサーバに割り当てた疑似乱数を比較し，SWP＞疑似乱数であれば該サーバを書き込み先サーバとする

0 番サーバの SWP は 1.0 であるため，必ず書き込み先サーバが選ばれる．データを書き込む方法はデバイスに依存する．

STEP3. サーバ番号が大きな順に書き込み先サーバの手前のサーバまで SRP とサーバに割り当てた疑似乱数を比較し，SRP＞疑似乱数であれば該サーバにデータ消去命令を送る

SRP は最も大きかったときの SWP の値であるため，書き込みサーバよりもサーバ番号が大きなサーバのうち，SRP＞疑似乱数となるサーバは古いデータを記憶している可能性がある．そのため，古いデータを消すためのデータ消去命令を送る．データを消去する方法はデバイスに依存する．

2.4 データ読み込み時のアルゴリズム

データ読み込み時には次のようなアルゴリズムによってデータを読み込むサーバを決定する．

STEP1. データ ID と各サーバのサーバ番号を元に 0.0 以上 1.0 未満の疑似乱数を生成し，各サーバに割り当てる

データ書き込み時と全く同じ方法で各サーバに疑似乱数を割り当てる．同じデータ ID であれば書き込み時も読み込み時も同じサーバには同じ疑似乱数が割り当てられる．

STEP2. サーバ番号が大きな順に SRP とサーバに割り当てた疑似乱数を比較し，SRP＞疑似乱数であればサーバからデータを読み込む．データを持つサーバからデータを読み込むまでこれを続ける

サーバからデータを読み込む際，必ずデータがサーバにあるわけではない．だが，ストレージにデータが書き込まれていれば必ず該アルゴリズムで選ばれるサーバのいずれかにデータがある．また，同じデータ ID を持つデータを複数回ストレージに書き込んだ場合，必ず最後に書き込んだデータを読み込むことが可能である．データを読み込む方法はデバイスに依存する．

2.5 データの冗長化

スケールアウト型ストレージではサーバの故障に対処するため，データを冗長化することが多い．Sequential Checking では次のようにデータを冗長化する．

1．1 データにつき n データ書き込むとき，n 台のサーバをグループ化する
2．各グループを 1 台のサーバとして取り扱い，アルゴリズムを実行する
3．グループ内でデータを冗長化する

データを書き込む際は書き込み先グループを Sequential Checking によって決定し，グループ内のサーバでデータを冗長化する．データを読み込む際は読み込み先のグループを Sequential Checking で決定し，グループ内の動作しているサーバから読み込む．

3. Sequential Checking の評価

この章では Sequential Checking の性質について定量的な評価を行う．Sequential Checking では疑似乱数を生成するアルゴリズムが非常に重要になるため，評価ではほぼ一様な疑似乱数を生成できるメルセンヌツイスターを用いた．3.1 節で基本的な性質を確認し，3.2 節で Sequential Checking の定量的な評価を行う．

3.1 基本的な性質の確認

本節では Sequential Checking の基本的な性質を確認する．本節で確認する基本的な性質は次の 2 点である．

1．サーバの空き容量とデータの書き込み回数がほぼ比例すること
2．サーバの空き容量を増減させても最新のデータを確実に読み込めること

第 1 にサーバの空き容量とデータの書き込み回数がほぼ

表 1：サーバ容量固定時のサーバの空き容量とデータ書き込み回数（データ量）

サーバ番号	SWP	データ量（TB）
0	1.000	99.683
1	0.500	100.295
2	0.333	100.119
3	0.250	99.596
4	0.200	100.142
5	0.167	100.165

比例することを確認した．まず，サーバ容量が固定の場合について確認した. サーバ台数を 6 台, サーバ容量を 100TB とし, データの大きさを 1GB とした. 書き込むデータの ID はユニークとし，データの量は 600TB とした．一部のサーバで書き込むデータの量がサーバ容量を超えることは無視する．このときの各サーバの SWP 小数点以下 3 位までおよび各サーバに書き込まれたデータ量を表 1 に示す．表 1 から明らかなようにデータ量（＝データ書き込み回数）はサーバの空き容量にほぼ比例する．

次にサーバ空き容量が変化する場合について調べた．サーバの空き容量は 0.0 以上 1.0 未満の一様乱数によって決定するとし，初期のサーバ台数を 2 台，2 回空き容量を決定しデータを書き込んだ後にサーバを 1 台ずつ追加し，最終的にサーバが 6 台になるとした．サーバに書き込むデータの ID はユニークとし，各書き込みの際に書き込むデータ数はサーバの空き容量の合計に 100000 をかけた数とした. このサーバの空き容量決定とデータの書き込みを 10 回行い，サーバの空き容量の合計（小数点以下 3 桁）と書き込まれたデータ数の合計を得た．参考に最終的な SRP と SWP（小数点以下 2 桁）を加えて，結果を表 2 に示す.

表 2：サーバ容量可変時の SRP，SWP，
サーバの空き容量の合計とデータ書き込み回数

サーバ番号	SRP	SWP	サーバ空き容量の合計	データ書き込み回数の合計
0	1.00	1.00	5.695	567632
1	0.77	0.41	5.611	561938
2	0.73	0.16	3.298	331479
3	0.29	0.03	1.660	165652
4	0.40	0.19	2.031	202758
5	0.26	0.26	1.021	102160

表 2 から明らかなようにサーバの空き容量の合計と書き込まれたデータ数はほぼ比例する．つまり，Sequential Checking によって，サーバの空き容量に比例した数，データを書き込むことができる.

第 2 にサーバの空き容量を増減させたとき，書き込んだ最新のデータを読み込めることを確認した．初期のサーバ台数を 1 台とし空き容量を決定し，データを書き込むたびに 1 台追加し最終的に 6 台になるとした．サーバの空き容量は 0.0 以上 1.0 未満の一様乱数によって決定した．サーバに書き込むデータ数は 1 回の書き込みで 1000000 データとした．データ ID は 0 から始まる整数とし，サーバ台数を 4 台に増やしたときにデータ ID を 0 にリセットした．つまり，サーバが 1 台から 3 台の時とサーバが 4 台から 6 台の時に同じデータ ID のデータを書き込む．データの書き込みが終わると，データを読み込み，後に書いたデータが読み込めることを確認した.

表 3：データを読み込めることの確認

サーバ番号	1 回目	2 回目	読み込み		RAND
	SWP	SWP	SRP	SWP	
0	1.00	1.00	1.00	1.00	0.49
1	0.73	0.46	0.77	0.77	0.89
2		0.15	0.68	0.68	0.02
3		0.38	0.38	0.17	1.00
4		0.22	0.22	0.06	0.09
5			0.20	0.20	0.59

表 3 に例としてデータ ID1234567 のデータを 1 回目に書き込んだとき, 2 回目に書き込んだときの SWP と読み込んだときの SRP 及び SWP と疑似乱数（RAND）を示す．読み込み先，書き込み先として選ばれる可能性のあるサーバの SRP もしくは SWP には下線を引いた．サーバ 3 の疑似乱数が 1.00 になっているのは四捨五入のためである．実験の結果，全ての最新のデータを読み込むことができる事がわかった.

3.2 定量的な評価

この節では Sequential Checking の定量的な評価を行う．第 1 にサーバの空き容量と書き込まれるデータ数の誤差を評価する．第 2 に実際の使用を想定した前提の元でデータを読み込む際にアクセスする必要があるサーバ数を評価する．第 3 に削除命令が非常に多くなる場合にデータを書き込む際に削除命令を送るサーバ数を評価する．最後に実機を用いて実際にデータの読み書きを行い，デファクトスタンダードのデータ分散アルゴリズムである Consistent Hashing [10]とアクセス速度の違いを評価する.

3.2.1 サーバの空き容量と書き込まれるデータ数の誤差

まず，サーバの空き容量と書き込まれるデータ数の誤差を評価する．サーバの台数は 256 台とし，各サーバの空き容量は 0.5 以上 1.5 未満の一様乱数によって決定する．それぞれの評価でユニークな ID を持つデータを（各サーバの空き容量の合計）×1000000 データ書き込み，空き容量から想定される書き込まれるデータ数と実際に書き込まれたデータ数の誤差を計算し, 各評価で最も大きな誤差を取る．評価は 100 回行った．評価の結果，データ数の最大誤差は 0.28%から 0.51%となり，平均で 0.35%となった．空き容量に対する書き込まれるデータ数の誤差が大きなとき，他のサーバの容量に余裕がある場合でも，あるサーバの容量が一杯になり，ストレージにそれ以上データを書き込めなくなる．空き容量に対する書き込まれるデータ数の誤差が小さいときストレージの容量を有効に利用できる.

3.2.2 データ読み込み時のアクセスサーバ数

次に，データを読み込む際にアクセスする必要があるサーバ数を評価する．Sequential Checking を用いたストレー

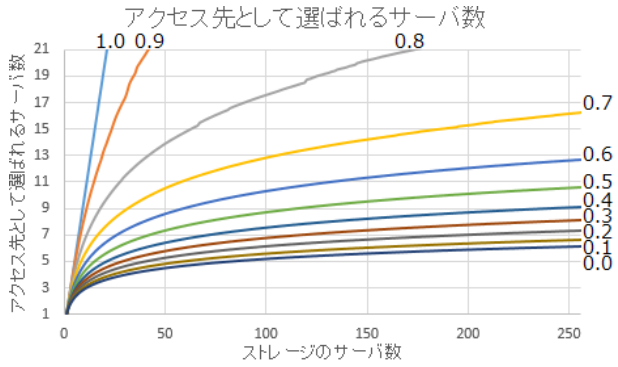
ジの使用法の想定として，一定の割合ストレージにデータが書き込まれたとき，ストレージの容量を増やす場合を想定する.そして,ストレージの容量が最大容量に達した後，書き込んだデータの総量がストレージの容量になるまでデータを書き込むと想定する．一部のサーバでサーバの容量を超える量のデータが書き込まれることは無視する．データの書き込みが終わった後，データを読み込むために必要なアクセスするサーバ数を評価の対象とする.

サーバ 1 台，サーバの容量 100TB を初期設定とする．サーバの最大容量は 1PB とし，1 回のストレージ容量の追加毎に最後に追加したサーバに 100TB 追加する．容量を追加しているサーバが最大容量に達すると次のサーバを追加する．ストレージの容量が N×100%（N＝0.0～1.0 まで 0.1 刻み）埋まる毎にストレージ容量を追加する．データのサイズは 1GB とする．サーバの最大台数は 1 台から 256 台まで評価する．ストレージの容量が最大容量に達した後書き込んだデータは必ず 1 アクセスで読み込むことが可能であるため，評価対象として次の 3 種類の結果を評価する.

1．読み込み先サーバ（アクセス先）として選ばれる可能性のあるサーバ数
2．データを読み込む際にアクセスしたサーバ数
（ストレージ容量を追加できなくなる前に書き込んだデータのみ）
3．データを読み込む際にアクセスしたサーバ数
（全てのデータ）

3.2.2.1. 読み込み先サーバ数

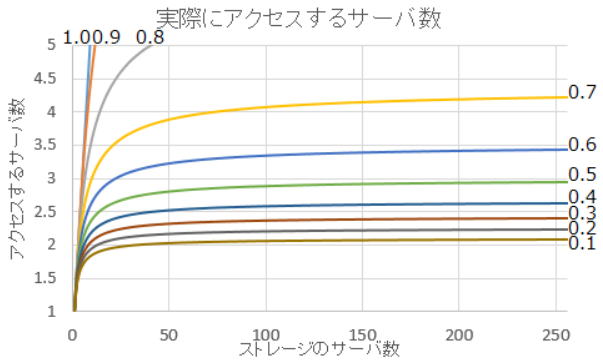
読み込み先サーバとして選ばれる可能性のあるサーバ数はグラフ 1 のようになった．N が小さい場合の結果の違いを示すため，N が 0.8 以上の場合の結果はグラフに収まっていない．例として N が 0.5 の時，つまり，ストレージが半分埋まるたびに容量を追加する場合について述べると，256 サーバある場合でもアクセス先として選ばれるサーバ数は高々11 サーバに過ぎないことがわかる.



グラフ 1：アクセス先として選ばれるサーバ数

3.2.2.2. アクセスサーバ数（容量を追加できなくなる前に書き込んだデータ）

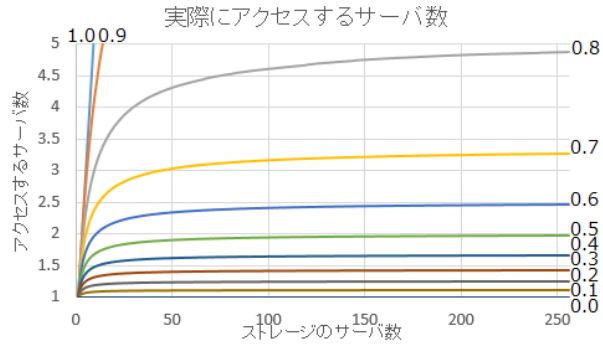
ストレージ容量を追加できなくなる前に書き込んだデータを読み込む際に実際にアクセスする必要があるサーバ数はグラフ 2 のようになった．N が小さい場合の結果の違いを示すため，N が 0.8 以上の場合の結果はグラフに収まっていない．同様に例として N が 0.5 の時について述べると，256 サーバある場合でもアクセスする必要があるサーバ数は平均 3 サーバ弱に過ぎないことがわかる．また，最大サーバ数が 30 を超えるとアクセスする必要があるサーバはほとんど増えないこともわかる．これは Sequential Checking の顕著なスケーラビリティを表す.



グラフ 2：容量を追加できなくなる前に書き込んだデータを読み込む際に実際にアクセスするサーバ数

3.2.2.3. アクセスサーバ数（全てのデータ）

ストレージに書き込まれた全てのデータを読み込む際に実際にアクセスする必要があるサーバ数はグラフ 3 のようになった．N が小さい場合の結果の違いを示すため，N が 0.9 以上の場合の結果はグラフに収まっていない．同様に例として N が 0.5 の時について述べると，256 サーバある場合でも平均 2 サーバ弱にアクセスすることによりデータを読み込むことが可能である.



グラフ 3：最大容量まで書き込んだデータを読み込む際に実際にアクセスするサーバ数

グラフ 1 から 3 から明らかなようにストレージに書き込まれたデータが少ないときにストレージの容量を追加した方がデータにアクセスするためにアクセスする必要があるサーバ数が少なくなる．だが，書き込まれたデータが少ないときに容量を追加する場合，コストが増大する．そのため，実際の使用に当たっては性能とコストのバランスを取って，システムを設計する必要がある.

3.2.3 削除命令数

さらに，極端に多くの削除命令を送る場合を想定した前提の元でデータを書き込む際に削除命令を送るサーバ数を評価する．Sequential Checking を利用するストレージを，容量が足りなくなったときサーバ及びサーバの容量を追加する通常想定されるような使用法で使った場合，削除命令をほとんど発行しない．そのため，極端に削除命令を発行するケースを人為的に作り出し，この極端に削除命令を発行する例での削除命令の発行数を評価する．サーバの台数を 8, 16, 32, 64, 128, 256 とし固定する．サーバの容量の増減が 100 回あるとし，各サーバの空き容量を 0.0 以上 1.0 未満の一様乱数を使って設定し，SRP と SWP を求める作業を 100 回繰り返した．そして，最終的な SRP と SWP を用いてデータをストレージに書き込み，この際の削除命令の数を測定した．これを 100 回繰り返し，平均を取った．SRP は SWP の最大値であり，SRP が大きく，SWP が小さいほど削除命令は多くなるため，このシミュレーションから計測される削除命令の数は実際の使用時の数と比較し，非常に多い数になる．結果は表 4 のようになった．非常に多くの削除命令を発行するように設定したにもかかわらず削除命令は平均で 1 を少し超える程度の数であることがわかった．実際に削除するデータ数はこれより遙かに少なく，データの削除がストレージの容量に与える影響は少ない．また，サーバ数が増えた場合でも削除命令数が増えることはなく，サーバ数 16 程度をピークにむしろ減ることがわかった．これは，サーバ数が増えるに従って同じサーバに書き込まれるデータであれば削除命令の数も増えるが，同時にそのサーバにデータを書き込む際に必要な削除命令数の数が少ないサーバに書き込まれるデータの割合も増え，サーバ数 16 前後を境に前者の効果よりも後者の効果の方が大きくなるためである．

表 4：削除命令の平均数

サーバ数	削除命令の平均数
8	1.283
16	1.343
32	1.292
64	1.227
128	1.162
256	1.103

3.2.4 実機の評価

最後に Sequential Checking を用いたストレージでの性能の低下を通常のストレージと比較するために，実機を用いてアクセスに必要な時間を評価した．クライアントサーバから同じハブに接続している 8 台のストレージサーバの memcached にデータの書き込み及び読み込みを行い，必要な時間を計測した．用いた機材などは次のようになる．

サーバ

CPU：INTEL Xeon X5550 2.67GHz，メモリ：24GB

ネットワーク：1000BASE-T

サーバ　クライアント：1 台，ストレージ：8 台

データ

データ　サイズ：1KB，ID：0 から順に整数

数：8,000,000 データ

クライアントソフトウエア

OS：CentOS 6.3，使用ライブラリ：libmemcached 1.0.18

データ分散アルゴリズム：Consistent Hashing(Virtual Server 100)，Sequential Checking

ストレージソフトウエア

OS：CentOS 6.3

memcached：1.4.4，使用オプション：-d –m 4096 –p 10000

ストレージとして memcached を用いており，Sequential Checking が想定しているテープデバイスや光学デバイスと比較し，分散アルゴリズムの実行，命令やデータのネットワーク転送，データの存在チェック，データの削除などの時間の比重が重くなり，ストレージサーバでのデバイスからのデータの読み書きの時間の比重が軽くなる．つまり，Sequential Checking が非常に不利になる環境である．

ベンチマークとして次のテストをした．

1．Consistent Hashing を用いて，ストレージサーバにデータを全て書く時間を測定した
2．Consistent Hashing を用いて，ストレージサーバに書いたデータを全て読む時間を測定した
3．Sequential Checking を用いて，ストレージサーバにデータを全て書く時間を測定した
4．Sequential Checking を用いて，ストレージサーバを 1 台としてデータを書き，以降データを 500,000 個書く毎にストレージサーバを追加し，ストレージサーバが 8 台になったときデータを 500,000+4,000,000 個書いた上で，ストレージサーバに書かれたデータを読み，データを読む時間を測定した

1 と 2 は通常の Consistent Hashing の書き込みと読み込みであり，3 は通常の Sequential Checking の書き込みを想定している．4 は Sequential Checking の通常の使い方として想定される，ストレージの容量が足りなくなるたびにサーバを追加する使い方を想定し，データを読み込んでいる．各テストでテスト対象になるデータ数は同じである．Sequential Checking を通常の使い方で使っている場合，削除命令を多数発行するケースは発生しないため，実機によるベンチマークでは削除命令を多数発行するケースについては測定していない．それぞれのテストを 5 回繰り返し，平均を取っている．結果は表 5 のようになった．

Consistent Hashing 書き込みと Sequential Checking 書き込みではサーバにアクセスする回数は非常に少ない削除命令の発行を除いて同じであり，分散アルゴリズムの実行時間

表 5：実機テスト

テスト	実行時間（s）
Consistent Hashing 書き込み	1345.3
Consistent Hashing 読み込み	1253.4
Sequential Checking 書き込み	1321.8
Sequential Checking 読み込み	1649.5

のみ異なる．実験の結果，実行時間の違いはほぼ誤差の範囲だった．Sequential Checking 読み込みでは 8,000,000 個のデータの読み込みに対して，約 11,400,000 回の読み込み命令を発行しており，8,000,000 個のデータ読み込みに対して 8,000,000 回の読み込み命令を発行する Consistent Hashing と比較し，3 割程度遅くなっている．なお，Consistent Hashing は最大 23.3%の各サーバのデータ数の誤差が発生したが，Sequential Checking の各サーバのデータ数の誤差は 0.2%に満たない．つまり，Consistent Hashing と比較し，Sequential Checking の方が遙かにストレージ容量を有効に使うことができる．

実機によるベンチマークから，小規模のストレージであれば，Consistent Hashing と比較し Sequential Checking はアルゴリズムの実行時間はほとんど変わらず，オーバーヘッドはアクセス命令の発行数の増加のみであることがわかる．

4. 議論

4.1 サーバの容量と比例する対象がデータを書き込む数であることについての議論

Sequential Checking はデータを消したとき，データを消したことにより発生する空き容量を利用することが難しい，もしくは不可能なデバイスを対象にしている．そのため，データを書き換える場合，元のデータを消して空き容量を再利用することは考えない．元データを消し空き容量を再利用することを考えるのであればデバイスを考え直すべきである．

そのため，十分な数のデータが存在し，大数の法則により各サーバが持つデータの平均サイズがほぼ等しくなると仮定したとき，データを書き込む回数とサーバが記憶しているデータ量はほぼ比例する．つまり，サーバの容量とデータを書き込む回数が比例することにより，サーバの容量を使い切ることができる．大数の法則は他のデータ分散アルゴリズム[10][11][14]においても前提となっている法則である．

4.2 サーバにより読み込み頻度が異なることについての議論

Sequential Checking ではデータを読み込む際，後に追加したサーバは記憶しているデータの数と比較し，データの読み込み頻度が高くなる．また，0 番サーバは常に読み込み先の候補として選ばれる．

後に追加したサーバが記憶しているデータの数と比較し読み込み頻度が高くなる問題は，該サーバが読み込むデータを記憶しているか否か判定する頻度が高くなる問題を引き起こす．これはサーバが記憶しているデータのデータ ID を Bloom Filter によって記憶することにより処理の重さを軽減することができる．

0 番サーバは読み込み先の候補として確実に選ばれるが，実際に 0 番サーバからデータを読み込む場合は該サーバがデータを持っている場合もしくはストレージがデータを持っていない場合に限られるため，アクセスの集中などの問題は発生しない．

4.3 サーバの削除及びすでにデータが書き込まれたデバイスの削除についての議論

Sequential Checking ではサーバの空き容量の削除は可能だが，サーバの削除及びすでにデータが書き込まれたデバイスの削除は通常できない．

だが，サーバの削除であれば，削除するサーバの SRP と SWP を 0.0 とし，削除するサーバに書き込まれたデータを再分散することにより実現できる．

データが書き込まれたデバイスの削除も同様に，デバイスの削除を考慮して SRP と SWP を設定し，デバイスに書き込まれたデータを再分散することにより実現できる．

だが，これらは Sequential Checking の特徴である，データを再分散しないデータ分散アルゴリズムという主張を弱めるため，可能であることを指摘するだけにとどめる．

4.4 アクセスサーバ数が極端に多くなる場合がありレイテンシが安定しないことについての議論

Sequential Checking では多くの場合で読み込み，書き込みともアクセスするサーバ数は少ない．だが，少数のケースで多数のサーバにアクセスする可能性があり，この場合，レイテンシが極端に長くなる．アルゴリズムの性質上この現象を防ぐことはできないが，Sequential Checking が想定しているデバイスはテープデバイスや光学デバイスであり，通常短いレイテンシを求められないコールドデータが記憶されるデバイスである．そのため，レイテンシが長くなっても問題にはならない．

Sequential Checking を RAM，HDD や SSD などといった短いレイテンシが求められるデータを記憶するデバイスを使用するストレージに用いる場合，レイテンシが大きく変動する可能性があることは留意しなければならない．

5. 関連研究

データをアルゴリズムによって記憶装置に分散配置するシステムは大きく分けて 2 種類ある．RAID と分散ストレージである．RAID は今日においても盛んに研究されている[6] [7] [8]が，Sequential Checking との関連性は薄い．そのため，ここでは分散ストレージ向けのデータ分散アルゴリズムについて紹介する．また，サーバ構成変更時にデータを移動しないデータ分散アルゴリズムの利用先として考え

られる，大容量データ向け光学ストレージを紹介する．

まず, Sequential Checking と類似する RUSHp を紹介する．そして，サーバ構成変更時にデータを移動しないデータ分散アルゴリズムは私の知る限り Sequential Checking の他に存在しないため，サーバ構成変更時に最小限のデータのみ移動するデータ分散アルゴリズムを紹介する．

RUSHp [9]は Sequential Checking と似たデータ分散アルゴリズムである．だが，RUSHp はサーバ追加時に最小限のデータのみ移動するアルゴリズムとして定義されており，アルゴリズム設計が Sequential Checking と異なる．

Consistent Hashing [10]はスケールアウトストレージ向け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．データをサーバにほぼ一様に分散することができ，サーバ構成変更時に最小限のデータ移動のみ要求する．ハッシュリングにサーバとサーバのバーチャルサーバを配置し，配置した位置から一定の方向に次のサーバまでの領域をそのサーバの領域とする．データアクセス時にはハッシュリング上にデータを配置し，配置した位置を領域とするサーバにアクセスする．Random Slicing [11]は Consistent Hashing の亜種であり，ハッシュラインをサーバの容量に合わせて分割するようにサーバをハッシュラインに配置することにより，Consistent Hashing よりも正確にサーバの容量に比例した数のデータをサーバに記憶できる．

Highest Random Weight [12]はサーバに一様にデータを分散することができ，サーバ構成変更時に最小限のデータ移動のみ要求する．データ ID と各サーバの ID を組み合わせたものをハッシュし，最もハッシュが大きなサーバにデータを記憶する. Highest Random Weight の亜種として CRUSH の Straw Bucket [13]や Weighted Rendezvous Hashing[14]があり，サーバ容量の違いに対応する．

大容量データ向け光学ストレージとしては Facebook とパナソニックの freeze-ray [15]が挙げられる．

6. まとめ

サーバ構成変更時にデータの移動が難しい又は不可能なデバイスを用いた分散ストレージに適用可能な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 Sequential Checking を紹介した．Sequential Checking はストレージにサーバを追加するときもしくはサーバの空き容量を増減するとき，すでに書き込まれたデータを移動することなく，サーバの容量に合わせた数のデータを書き込むことができ，一意にデータを書き込むサーバを決定することができ，データを記憶したサーバを必ず含む多くの場合で少数のサーバを判定可能である. シミュレーションにより，Sequential Checking の動作を確認し，特性について評価を行った．その結果，設定した環境では 256 サーバあるとき平均 1.98 サーバにアクセスすればデータを読み込めることが分かった．Sequential Checking により従来では不可能だったテープデバイスや光学デバイスを用いたスケールアウト型分散ストレージが可能になる．

参考文献

[1] 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. DOI=http://dx.doi.org/10.1145/1294261.1294281

[2] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” SIGOPS Oper. Syst. Rev. 44, 2, April 2010, pp. 35-40. DOI=http://dx.doi.org/10.1145/1773912.1773922

[3] 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. DOI=http://dx.doi.org/10.1145/272991.272995

[4] M. Saito and M. Matsumoto, “SIMD-Oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator,” *Monte Carlo and Quasi-Monte Carlo Methods*, Springer Berlin Heidelberg, pp. 607-622. http://dx.doi.org/10.1007/978-3-540-74496-2_36

[5] G. Marsaglia, “Xorshift rngs,” *Journal of Statistical Software*, 8.14, 2003, pp. 1-6.

[6] I. Iliadis and V. Venkatesan, “Rebuttal to “Beyond MTTDL: A Closed-Form RAID-6 Reliability Equation,” *Trans. Storage*, 11, 2, Article 9, March 2015, 10 pages. DOI=http://dx.doi.org/10.1145/2700311

[7] E. Lee, Y. Oh, and D. Lee. “SSD caching to overcome small write problem of disk-based RAID in enterprise environments,” *In Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC 2015, New York, NY, USA, pp. 2047-2053. DOI=http://dx.doi.org/10.1145/2695664.2695886

[8] G. Zhang, K. Li, J. Wang and W. Zheng, “Accelerate RDP RAID-6 Scaling by Reducing Disk I/Os and XOR Operations,” *Computers, IEEE Transactions on*, Volume:64 , Issue: 1. pp. 32-44. DOI:10.1109/TC.2013.210

[9] Honicky, R. J., and Ethan L. Miller. "A fast algorithm for online placement and reorganization of replicated data." *Parallel and Distributed Processing Symposium*, 2003. Proceedings. International. IEEE, 2003.

[10] 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. DOI=http://dx.doi.org/10.1145/258533.258660

[11] 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.

[12] 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.

[13] 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 . DOI=http://dx.doi.org/10.1145/1188455.1188582

[14] http://www.snia.org/sites/default/files/SDC15_presentations/dist_sys/Jason_Resch_New_Consistent_Hashings_Rev.pdf

[15] https://panasonic.net/avc/archiver/freeze-ray/