東海大学紀要情報通信学部
Vol.11,No.2,2018,pp.1-7

論文

スマートデバイスとサーバによるイメージデータ保存システム
—ハッシュチェーンによる楽観的検査プロトコルを用いた保存と復旧—

童 磊[*1]，小林　洋[*2]

Image Data Storage System Using Smart Devices and Server
- Storage and Recovery Using Optimistic Checking Protocol by means of Hash Chain -

by

Lei TONG[*1] and Hiromi KOBAYASHI[*2]

(received on Aug.24, 2018　&　accepted on Nov.15, 2018)

あらまし

最近，文書や音声等の記録をスマートフォンやタブレット等のスマートデバイスでイメージデータとして保存するような機会が増えて来ている．そこで，本研究では，重要なイメージデータをスマートデバイス経由でサーバのデータベースに保存するシステムの提案と開発を行う．本システムの特徴は，スマートデバイスとサーバのデータ間にハッシュチェーン（hash chain）を構成し，これを用いて楽観的（optimistic）な軽量の検査プロトコルにより，一方のデータが書き変わったり失われたような場合に，他方のデータを用いて復旧できる点にある．

Abstract

Nowadays, the use of image data as records of documents or speech in smart devices such as smartphones or tablets is increased. This paper addresses the development of a system that preserves important image data in a database server via smart devices. The feature of this system is the usage of an optimistic light weight protocol by means of a hash chain that constructed using data between each smart device and this server. Faulty data in one side can be recovered from those in the other side using this protocol.

キーワード: ハッシュチェーン，楽観的検査プロトコル，イメージデータ，復旧，スマートフォン
Keywords: *hash chain, optimistic checking protocol, image data, recovery, smartphone*

1. はじめに

近年，センサ技術の発達により，AndroidやiOSのスマートフォンやタブレット等のスマートデバイスで，画像等のイメージデータが簡単に収集できるようになって来ており，文書や音声等の記録をタブレット等で手軽にイメージデータとして記録するような状況も増えて来ている．スマートデバイスでは，リレーショナルデータベース（RDB）がライブラリとして簡単に用いる事ができるようになっているため，取得したイメージデータが保存すべき重要な記録である場合，一旦，スマートデバイスのRDBに保存し，後でサーバのRDBにまとめて送り保存することが考えられる．両者がRDB同士であれば，整合性の維持が比較的容易なのではないかと考えられる．そこで，先に我々は，この考えを基にイメージデータ保存システムを提案したが[1-2]，イメージデータのハッシュ化の方法や検査用のプロトコルの検討が不十分であった．そこで本稿では，複数の属性値付きのイメージデータのハッシュ化と検査・復旧用のプロトコルを実装用に詳細化したものを示す．このシステムの特徴は，スマートデバイスとサーバのRDBのデータ間で，属性値付きの複数のイメージデータのハッシュ値を基にハッシュチェーン（hash chain）を構成し，これを用いて定期的に楽観的（optimistic）（または，投機的（speculative））な検査プロトコルにより効率的な検査を行い，一方のデータの一部が，障害やセキュリティ上の理由により書き変わったり失われてしまったような場合には，その箇所をハッシュチェーンを利用して探索し，他方のデータを基に復旧できる事である．書き変わった箇所の探索の際には，二分探索法と線形探索法を基にした方法が考えられるので，本稿では，その処理効率の比較検討をシミュレーションにより行った結果について示す．実装については，要素技術的な部分については，検査や回復用のプロトコルを作り上げる際に確認のために個々に試作を行った．全体をまとめた開発については，次の段階として現在実施中であり，本稿の最後で，現在行っているAndroidタブレットとPCサーバによるシステムのプロトタイプ開発の概要について述べる．

*1 情報通信学研究科情報通信学専攻修士課程
Graduate School of Information and Telecommunication Engineering, Course of Information and Telecommunication Engineering, Master’s Program
*2 情報通信学部情報メディア学科 教授
School of Information and Telecommunication Engineering, Department of Information Media Technology, Professor

2. 関連研究

ハッシュチェーンの原理は Lamport らにより提案

され [3-5]，最近では複製（replica）のある分散データベースの高信頼化のための practical Byzantine fault(PBFT)プロトコル [6-9]の一種である Zyzzyva[9-12] 等で用いられている他，ビットコインシステム [13-15] でのブロックチェーンでは P2P 向けに複雑な構成にしたものが，過去の履歴データの改ざん防止のために用いられている．但し，本システムでは，ブロックチェーンのように P2P 向けに複雑にし，計算機資源を大量に消費するものではなく，クライアントサーバ向けの検査プロトコルにハッシュチェーンのみを用い，軽量なプロトコルとしている．

3. システムの構成と処理の概要

本システムの構成を Fig.1 に示す.本システムは，イメージデータを取得するための複数台のタブレットからなるクライアントと，これらからイメージデータを収集するサーバで構成する．クライアントとサーバには各々RDB があり，サーバ側ではハッシュチェーンでのハッシュ値を保存するためのハッシュ値 DB（以降，HDB）を用いる．HDB は，場合によっては，別ディスクとするなど信頼性の高いアーキテクチャとすることも考えられるが，プロトタイプにおいては，サーバ DB と同一ディスク内に置くことにする．

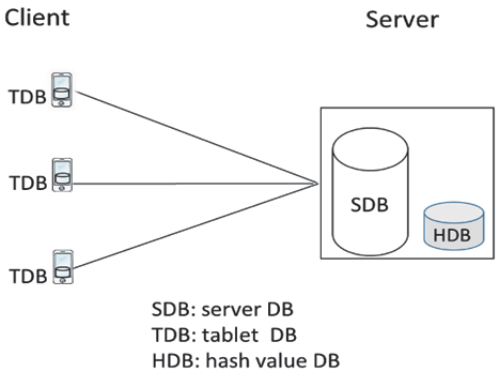


Fig.1 System architecture outline.

システムの処理手順としては，先ず，タブレット側でイメージデータ（とりあえずは，写真）を取得後，タブレット側の DB（以降，タブレット DB）に保存する．次に，定期的にタブレット DB の保存データをサーバ側の DB（以降，サーバ DB）に転送する．この手続きは次のようにして行われる．

［手続き 1　データ保存と転送］
① タブレット等で写真を撮影すると，写真が撮影用フォルダに格納される．
② フォルダに保存した写真を DB に格納する．イメージデータを RDB に保存する場合，blob 方法ではイメージデータをそのまま格納する．一方，link 方式では，イメージデータのリンクのみを保存し，写真は RDB の外部に保存する．このとき，4. で示す漸化式を用いたハッシュチェーンの計算を行う．
③ RDB に保存した写真を確認する場合には，写真を画面に表示させる．
④ 定期的に，写真をタブレット RDB からサーバに転送する．この時，blob 方式では，RDB からイメージデータを取り出してそのままサーバに転送する．link 方式では，まず RDB から写真のリンクを取り出して，それに基づき格納用フォルダから写真を取り出しサーバに転送する．
⑤ サーバ側では，タブレット等から取得したデータを，blob 方式ではそのままサーバ側の RDB に格納する．link 方式では，写真を受信用フォルダから格納用フォルダに保存後，写真のリンクを RDB に格納する．このとき，4. で示す漸化式を用いたハッシュチェーンの計算を行う．
⑥ RDB に保存した写真を確認する場合には，写真を画面に表示させる．

ところで，イメージデータの RDB への保存方法については，ANSI/ISO SQL92（SQL2）規格により直接 blob（binary large object）形式で表に保存すること（以降，blob 方式）も可能であるが，容量の問題からイメージデータへのリンクのみを表に保存しイメージデータ自体は RDB の外部に保存すること（以降，link 方式）も良く行われている．両者による実装の違いについてここで簡単に示す．イメージデータをタブレット RDB に保存する場合の表の定義の基本的な部分を Fig.2 に示す．Fig.2(a)は表にイメージデータをそのまま格納する blob 方式であり，(b)は表にイメージデータのリンクのみを保存しイメージデータ自体は RDB の外部に保存する link 方式での定義である．なお，pid はイメージデータの id，nid はタブレットの id を示している．各々の格納は Fig.3(a)，(b)のように行われる．

```
create table ImageData(
   pid    int not null auto_increment,
   image    blob,
   caption varchar(255),
   timestamp varchar(255),
   nid       varchar(255),
   primary key(pid)
 )
```

(a) blob

```
create table ImageLink(
   pid    int not null auto_increment,
   file_path varchar(255),
   caption varchar(255),
   timestamp varchar(255),
   nid       varchar(255),
   primary key(pid)
 )
```

(b) link

Fig.2 Definition of tables with image data.

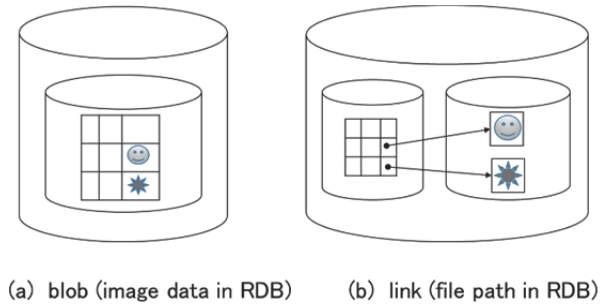


Fig.3 Two methods for storing image data.

4. ハッシュチェーンを用いたプロトコル

ハッシュチェーンは，基本的には $h_n=H(h_{n-1}, d)$ という漸化式でデータを繋いだものである．ここで，Hはハッシュ関数で，h_n と h_{n-1} は各々ステップ n と n-1 でのハッシュ値，d はデータの要約値を表し，h_n の系列はデータの履歴を表している．

今回のハッシュチェーンでは，複数枚の属性値付きのイメージデータからハッシュ値を求め，それを基にハッシュチェーンを構成するために，Fig.4 のような階層的な処理を行っている．

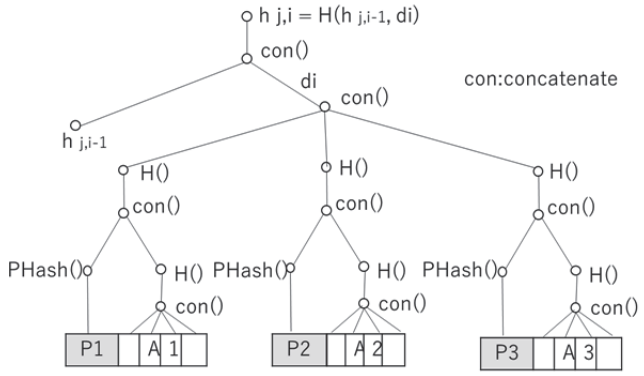


Fig.4 Creation of hash chain.

今，クライアント j で保存している複数枚のイメージデータをまとめてサーバに送るとする．以下は，blob 方式で示すが，link 方式でも同様に行える．クライアントからサーバへの送信回数を i とする．三枚の写真での例を示す．写真データ $P_{j,i}$ には，RDB に保存する際に，キャプション等の属性 $A_{j,i}$ が加えられるとする．すると，クライアント j の i 回目の送信データ $D_{j,i}$ は，写真が三枚の場合，$D_{j,i}((P_{j,i,1}, A_{j,i,1}), (P_{j,i,2}, A_{j,i,2}), (P_{j,i,3}, A_{j,i,3}))$ と表すことができる．このデータを基に，手続き 2 のように複数回ハッシュ化を行っている．

[手続き 2　ハッシュチェーンの作成]
①写真を一枚ずつ Perceptial hash 法（以下，PHash 法）[16]を用いて 64bit に圧縮する．
$h_{j,i,1}$=PHash($P_{j,i,1}$)，$h_{j,i,2}$= PHash($P_{j,i,2}$)，
$h_{j,i,3}$= PHash($P_{j,i,3}$)
②写真のキャプション等の属性値について，各属性値の文字列を結合後，FNV-1 アルゴリズム [17]を用いてハッシュ化を行う．
$k_{j,i,1}$=H($A_{j,i,1}$)，$k_{j,i,2}$= H($A_{j,i,2}$)，$k_{j,i,3}$= H($A_{j,i,3}$)
③ハッシュ化した各写真にハッシュ化した各属性値を結合して，各々FNV-1 アルゴリズムを用いてハッシュ化を行う．
$d_{j,i,1}$=H($h_{j,i,1}$，$k_{j,i,1}$)，$d_{j,i,2}$=H($h_{j,i,2}$，$k_{j,i,2}$)，
$d_{j,i,3}$=H($h_{j,i,3}$，$k_{j,i,3}$)
④これらの三枚分の値を一つに結合し，最後に，一つ前のハッシュ値と共にハッシュ化する．つまり，$d_{j,i}$=($d_{j,i,1}$，$d_{j,i,2}$ ，$d_{j,i,3}$)と結合によりまとめ，更に，一つ前のハッシュ値 $h_{j,\ i-1}$ と $d_{j,i}$ を結合後，FNV-1 アルゴリズムを用いてハッシュ化を行い，以下の漸化式の値を得る．
$h_{j,i}$=H($h_{j,i-1}$，$d_{j,i}$)　（但し，hj,0 は nonce（乱数））

手続き 3 にハッシュチェーンを用いた転送プロトコルを示す．手続き 3 は，m 台のクライアントとサーバで，各々独立した m 組のハッシュチェーンを用いた送受信プロトコルである．

[手続き 3　送受信プロトコル]
@client j (n_k: node j) step j　(j = 1, …, m)

while (1){

$d_{j,i} = H$ ($D_{j,i}$)　　//digest of data
$h_{j,i} = H(h_{j,i-1}, d_{j,i})$　　//summary of the history

send < $D_{j,i}$, nid> to n_0

i++
}

@server(n_0: node 0) step i

while (1){

receive < $D_{j,i}$, nid > from n_j　(j = 1, …, m)

$d_{0,j,i} = H$($D_{j,i}$)　　//digest of data
$h_{0,j,i} = H(h_{0,j,i-1}, d_{0,j,i})$　　//summary of the history

i++
}

このクライアント側の処理において，$D_{j,i}$ はステップ i（ターン i）でクライアント（ノード n_j (j = 1, …, m)）からサーバ n_0 に送られるデータである．$D_{j,i}$ は画像データ $P_{j,i}$ と属性 $A_{j,i}$(pid, cp, ts, i, nid) からなるデータで，pid, cp, ts, i, nid は各々，pid: 画像データ id, cp: キャプション，ts: タイムスタンプ，i: 送受信（ターン）回数，nid: ノード id を表している．$d_{j,i}$ は $d_{j,i} = H(D_{j,i})$から作られるデータの要約値であり，$h_{j,i}$ は $h_{j,i} = H(h_{j,i-1}, d_{j,i})$から作られる履歴の要約値を表している．

データ $\langle D_{j,i}$, nid $\rangle$ がサーバ n_0 に送られると，サーバ側の処理では，$D_{j,i}$ をステップ i でノード j から収集した増分データとすると，m 個のクライアントから送られるデータの和 $D_{0,i} = \Sigma D_{j,i} = D_{1,i} \cup D_{2,i} \cup$

…∪$D_{k,i}$ ∪…∪$D_{m,i}$ がステップ i でのサーバ側の増分データとなる．$d_{0,j,i}$ は $d_{0,j,i} = H(D_{j,i})$ から作られるデータの要約値，$h_{0,j,i}$ は $h_{0,j,i} = H(h_{0,j,i-1}, d_{0,j,i})$ から作られる履歴の要約値を表している．

HDB も考慮した，サーバとクライアント間のステップ i とステップ i+1 の間のシーケンス図を Fig.5 に示す．この場合，HDB では，クライアント j 毎にクライアント側のハッシュ値 hj,i とサーバ側のハッシュ値 h0,j,i の表が必要で，クライアントが m 台の場合，2m 列のハッシュ値を保存する表が必要になる．

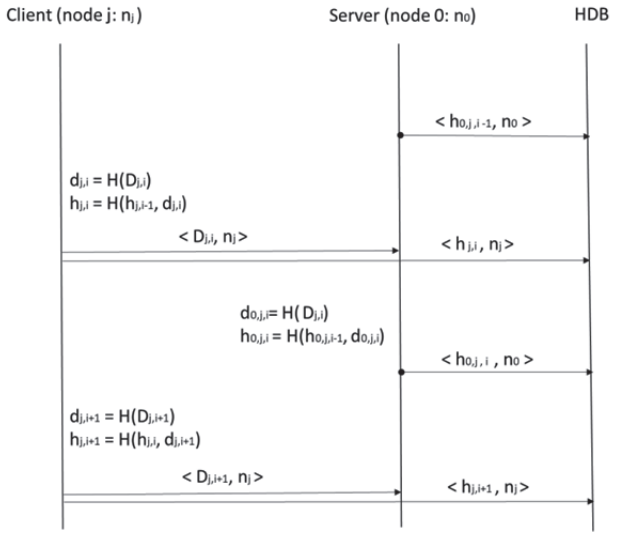


Fig.5 Sequence of hash chain protocol.

5. 検査プロトコル

5.1 検査と復旧の手続き

サーバ側でのハッシュ値の検査については，サーバ側で定期的に設定された区間でハッシュ値の再計算を行う．この際，Fig.6(a)に示すように，区間の始点 p から終点 q まで逐一再計算値と HDB からの値のマッチングを行うのが保守的（conservative）な検査手法なのであるが，ここでは，Fig.6(b)のように，区間の始点 p の後いきなり終点 q に飛んでマッチングを行うという楽観的（optimistic）な手法を取ることにする．楽観的な手法では，終点 q でハッシュ値に誤りが無い場合にはその区間内の画像を含むデータが正しいままであるために，これでこの区間の検査は終わりとなる．ハッシュ値に誤りがある場合には，その区間内のどこかで画像を含むデータが書き変わっている事になるのでその場合に限り誤りデータの箇所を特定するための処理を行う．誤りデータの箇所が特定出来たならば，クライアント側から該当する正しいデータを受領する事により復旧を行う．誤りデータが稀にしか無いような場合には，保守的な方法よりも楽観的な方法の方が処理効率が良い．サーバ側でのハッシュチェーンを用いた検査プロトコルと復旧方式を手続き 4 に示す．手続き 4 では，区間[p, q]でチェックする場合，まず p の箇所で HDB からハッシュ値 $h_{0,j,p-1}$ を受け取りハッシュ計算の漸化式により，q の箇所までハッシュ値の再計算を行い $h^*_{0,j,q}$ を得て，そこで HDB から受け取った $h_{0,j,q}$ との比較を行い，$h^*_{0,j,q}$ と $h_{0,j,q}$ の値が等しければ，その区間には誤りデータは存在しないので次の区間の検査に移る．つまり，誤りデータはめったに存在しないという前提の楽観的な手法である．しかし，$h^*_{0,j,q}$ と $h_{0,j,q}$ の値が等しくなければその場合に限り区間[p,q]でハッシュ値が一致しなくなった先頭の箇所をまず求め，その箇所のデータをクライアント側から再送してもらいデータの復旧を行う．この箇所でのハッシュ値が一致したら，ここを起点に q までのハッシュ値の計算を再度行いハッシュ値の照合を行う．つまり，複数個の誤りデータがある場合には，最初の誤りデータの箇所以降のハッシュ値が全て正しくないので，先頭から一つずつ誤りデータの箇所を特定し復旧していくことになる．

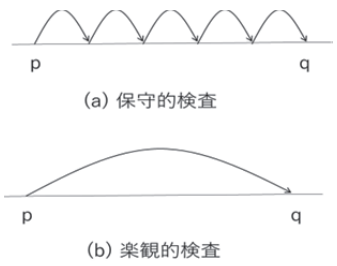


Fig.6 Conservative and optimistic checking.

[手続き 4　サーバ側の検査プロトコルと復旧]
① 区間[p, q]におけるクライアント j から受け取ったデータ $D_{j,p}$ から $D_{j,q}$ までのチェックを行うために，HDB から $h_{0,j,p-1}$ を受け取る．
② ハッシュ計算の漸化式を用いて p から q までハッシュ値の再計算を行い $h^*_{0,j,q}$ を得る．
③ HDB から $h_{0,j,q}$ を受け取り $h^*_{0,j,q}$ とのマッチングを行う．値が一致した時はこの区間のチェックを終了するが，値が一致しなかった時には誤りデータが存在する先頭の箇所を特定するために④へ．【楽観的手法】
④ 区間[p, q]で後述の手続き 4a（または 4b）を用いた探索で[p, q]で誤りデータが存在する先頭の箇所 q_f を特定する．
⑤ サーバは，クライアント j から誤りデータの箇所 q_f に該当するデータを受け取り復旧させる．
⑥ q_f の次の箇所を始点として残りの区間[q_f + 1, q]のチェックを行うため①へ．

④での探索手法として，二分探索法を基にした方法（binary search based method ：以降 BS 法）と線形探索法を基にした方法（sequential search based method：以降 SS 法）を以下に示す．Fig.7 は，誤りデータの箇所 m が複数個ある場合に，BS 法により，先頭の誤りデータの箇所を特定するまでの処理を示している．

［手続き 4a　BS 法］

以下では，区間[p,q]を[p_0，q_0]と記述することにし，k 番目に探索したハッシュ値の箇所を q_k とする．q_k では，HDB から該当する箇所のハッシュ値 $h_{0,j,k}$ を受領し，再計算値 $h^*_{0,j,k}$ との比較を行う．

k=0 から始め，1 回の処理ごとに，k+1→k とする．

① $h_{0,j,k} \neq h^*_{0,j,k}$ の場合，次の後退（backward）処理を行う．

$q_{k+1} = q_k - \square (q0 - p0)/ 2^{k+1} \square$（$\square\ \square$ は天井関数）

但し，$\square (q0 - p0)/ 2^{k+1} \square$=1 になったとき停止し，$q_{k+1}$ が誤った値の箇所である（$q_f = q_{k+1}$）．

② $h_{0,j,k} = h^*_{0,j,k}$ の場合，次の前進（forward）処理を行う．

$q_{k+1} = q_k + \square (q0 - p0)/ 2^{k+1} \square$（$\square\ \square$ は天井関数）

但し，$\square (q0 - p0)/ 2^{k+1} \square$=1 になったとき停止し，$q_{k+1}$ の次の箇所 q_{k+1} +1 が誤った値の箇所である（$q_f = q_{k+1} + 1$）．

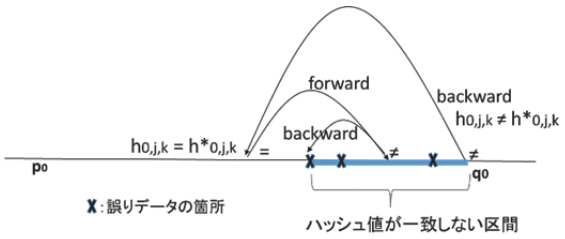


Fig.7 Process of forward and backward in BS method.

一方，q でのハッシュ値が一致しない場合，p の箇所から順次一つずつ比較チェックして行くのが，次に示す SS 法である．

［手続き 4b　SS 法］

区間[p_0，q_0]で，先頭から，一つずつ順次，ハッシュ値を前進方向に比較チエックしていき，

$h_{0,j,k} \neq h^*_{0,j,k}$ となる箇所 q_k を特定する．

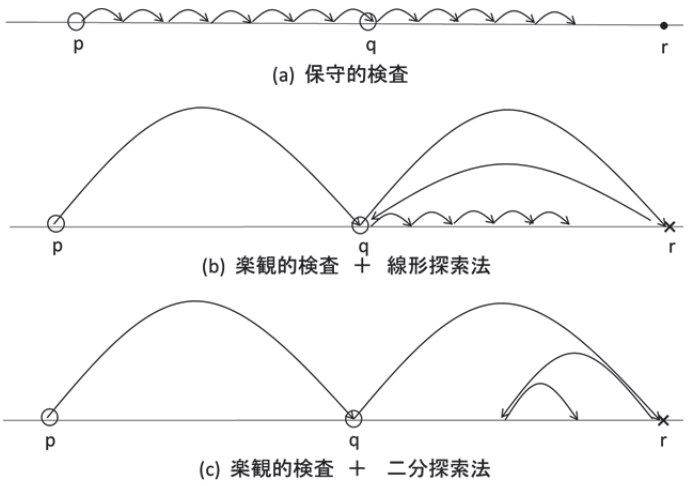


Fig.8 Process of conservative checking, and optimistic checking + binary search or sequential search.

以上に示した，保守的検査と楽観的検査，および楽観的検査での線形探索法と二分探索法についてまとめて図示したものを Fig.8 に示す．

5.2 二つの探索手法による検査プロトコルの比較評価

手続き 4 での二つの探索手法，BS 法と SS 法を用いた場合の検査プロトコルについて，処理効率の比較を行うために，Java によりシミュレーションプログラムを作成しシミュレーションを実行した．

まず，誤りデータが一か所，つまり m = 1 の場合には，区間数 N =1,2,4, 8, …, 128 の各々について，一様乱数により誤りデータの箇所を変化させたときの探索回数 n の 30 回の平均値をシミュレーションにより求めたところ Fig.9 のようになり，BS 法の方が SS 法より効率的であることが解る．なお，m=1 の場合については，解析的にも求めることができる．BS 法については，二分探索（binary search）が基になっており，区間[p,q]で区間数 N，誤りデータが一か所の場合の探索回数 n の最小値は n_{min} = 1 で，最大値は $N = 2^n + 1$ より $n_{max} = \log(N-1)$ で，平均値は O(log N)になる．一方，SS 法については，最小値 n_{min} = 1，最大値 n_{max} =N で，平均値は O(N)となり，Fig.9 のシミュレーション結果と一致する．

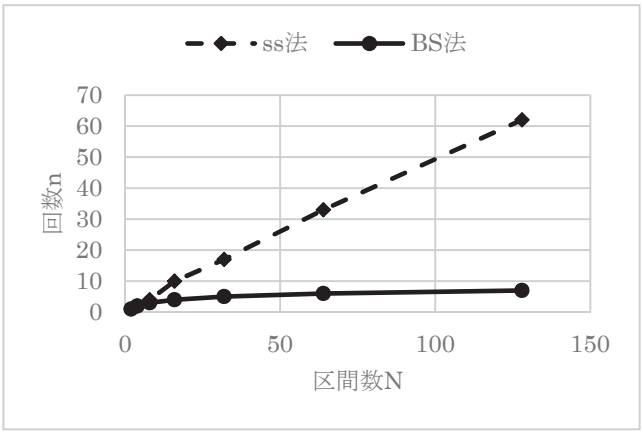
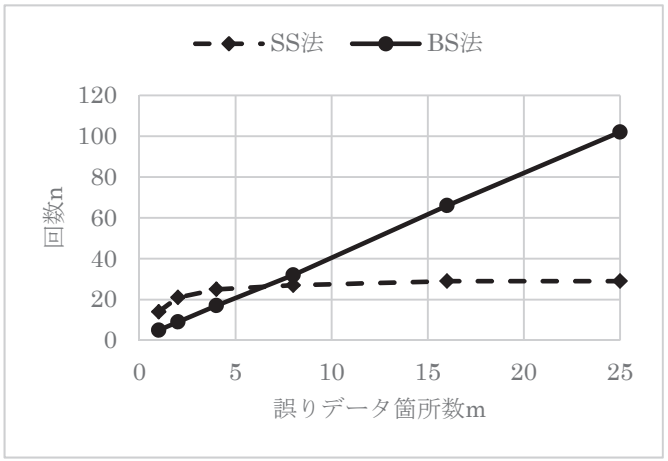


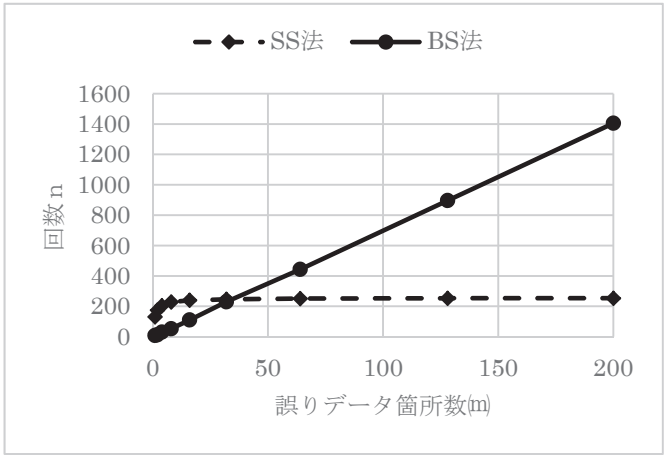
Fig.9 The number of search times varying the interval number N when the number of faulty data m=1.

次に，誤りデータの箇所が複数の m 個ある場合については，解析的に求めるのが困難で，シミュレーションにより求めることになる．このプログラムでは，誤りデータの箇所が m 個ある場合，探索によりまず先頭の誤りデータの箇所を特定し回復させ，残りの区間について誤りデータが無くなるまで，同様の処理を繰り返してその処理回数 n を求めている．区間数 N=32 の場合の，一様乱数により m 個（m = 1, 2, 3, …, 25）の誤りデータ箇所を作り出し，これらを全て先頭から一つずつ探索後，回復させるまでの回数 n の 30 回の平均値を求めた結果を Fig.10(a)に示す．すると，区間数 N に対して誤りデータ箇所数 m が小さいうちは BS 法が効率的だが，途中 m = 6.5 付近で BS 法と SS 法のグラフは交差し，それ以上 m が大きくなると SS 法の

方が効率的なのが解る．これは，区間数の値を変えても同様で，例えば，N=256 の場合には，Fig.10(b)のように m = 32.0 付近で BS 法と SS 法のグラフが交差した．ところで，この検査プロトコルは，区間内を逐次チェックしていくのではなく，HDB のハッシュ値との照合によるチェックは区間の始端の次は終端で行っており，終端のハッシュ値が誤っていた時のみ区間内の誤りデータの箇所を探索し復旧するという，誤りデータがめったに存在しないという前提の楽観的（optimistic）な手法として開発しているので，以上のシミュレーション結果から，探索手法としては BS 法の方が適していることが解る．



(a)　N=32



(b)　N = 256

Fig.10 The number of search times n when varying the number of faulty data m at the interval number N.

6. 実装

システムの開発については，サーバ側は，HP EliteDesk800 G1 SF/CT CPU: i7-4770 (3.4GHz)， RAM: 16GB　Windows10(64bit) に，XAMPP により Apache をインストール，RDB には MySQL[18]を用い，主に PHP により作成している．クライアント側は，タブレット端末である ASUS ZenPad3 8.0 (Android6.0)を用い，SQLite[19]と Java により作成を行っている．データ転送部分については，軽量的ウェブリクエストライブラリ okhttp3[20]を用いている。

まず，三つの要素技術の部分，①RDB へのイメージデータの二つの格納方式を用いた入力と検索部分，②データ転送部分，③サーバでのハッシュ化部分について各々のプロトタイプを作成した．現在，全体をまとめたプロトタイプを作成中である．

7. おわりに

スマートフォンやタブレット等のスマートデバイスで，文書や画像データを写真として取得したり，動画や音声データを取得することは，最近では良く行われている．これらのイメージデータの記録の中には，何らかの証拠として確実に保存しておきたいデータが含まれている場合がある．このような場合に，重要なデータはスマートデバイスからサーバに転送して保存しておくことが考えられる．このようなデータ転送は，オンラインでリアルタイムに行うのが理想的なのだろうが，現状では回線容量やデータベースサーバの処理能力による制約のため，回線容量が十分にある場所で，サーバに負荷がかかっていない時間帯に一括して行うという方式が現実的なのではないかと思われる．そこで，重要なイメージデータは，一旦，スマートデバイスの RDB に保存した後，サーバの RDB に転送する本システムを提案した．本システムで提案したデータ転送プロトコルでは，スマートデバイスとサーバのデータ間で，複数の属性値付きのイメージデータをまとめたハッシュ値の漸化式を用いてハッシュチェーンを構成することにより，一方のデータの一部が書き変えられたり失われた場合に，ハッシュチェーンを用いた楽観的検査プロトコルによりその箇所を特定し，他方のデータを用いて復旧することが可能なようにしている．このプロトコル内で用いる誤りデータ箇所の探索方法については，シミュレーションにより定量的評価を行った結果，二分探索を基にした BS 法が適していることが解った．ハッシュチェーンというと最近では，ブロックチェーンで知られているが，ブロックチェーンは P2P 向けに複雑な構成にしたもので，大量に計算機資源を消費することが問題になっている．本稿で示したプロトコルは，ハッシュチェーンのみを用いたクライアントサーバ向けの軽量なものである．本研究では，要素技術となる各部分ごとに実装を行いながら研究を進めた後，シミュレーションにより検査プロトコルの評価を行い，現在は，全体を実装したプロトタイプを作成中である．また，ハッシュチェーンの効率的な用い方や復旧方式の改善についても更なる検討が必要と思われ，これらが今後の課題と考えている．

参考文献

1) L.Tong and H.Kobayashi, Data Collection System Using Smart Device Databases and a Hash Chain Protocol, Lecture Notes on Information Theory, Vol. 5, No. 1, pp.13-16, 2017.
2) 童磊, 小林洋, タブレットDBとハッシュチェーンを用いたイメージデータ収集システム, 情報処理学会第 80 回全国大会講演論文集, 第 1 分冊, pp.265-266, 2018.
3) L. Lamport, Password Authentication with Insecure Communication, CACM 24, 11, pp. 770-772, 1981.
4) Y.-W. Peng, and W.-M. Chen, A Recursive Algorithm for General Hash Chain Traversal, Proc. IEEE 17th CSE, pp.1171-1174, 2014.
5) S. Bittl, Efficient construction of infinite length hash chains with perfect forward secrecy using two independent hash functions, Proc. 11th SECPYPT, pp.1-8, 2014.
6) M. Castro, and B. Liskov, Practical byzantine fault tolerance, OSDI'99, pp.173-186.
7) M. Castro, and B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Trans. on Computer Systems,20, 4, pp.398-461,2002.
8) M. Castro, Practical Byzantine Fault Tolerance. PhD thesis, MIT, Jan. 2001.
9) W. Thao, Building Dependable Distributed Systems, Wiley, 2014.
10) R. Kotla, A. Clement, E. Wong, L. Alvisi, and M. Dahlin, Zyzzyva: Speculative Byzantine Fault Tolerance, CACM, 51, 11, pp. 86-95, 2008.
11) R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, Zyzzyva: Speculative Byzantine Fault Tolerance, ACM Trans. on Computer Systems, 27, 4, pp. 1-39, 2009.
12) Y. Matsumoto and H. Kobayashi, A speculative Byzantine algorithm for P2P system, Proc. IEEE 2010 PRDC, pp. 231-234, 2010.
13) S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf.
14) E. Anceaume, T. Lajoie-Mazenc, R. Ludinard, and B. Sericola, Safety analysis of Bitcoin improvement proposals, 2016 IEEE 15th NCA, pp.318-325, 2016.
15) G. Hurlburt, Might the Blockchain Outlive Bitcoin?, IT Professional, 18, 2, pp.12-16, 2016.
16) N.Krawetz, http://www.hackerfactor.com/blog/index.php?/archives432-Looks-Like-It.html
17) T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C.Stein, Introduction to Algorithms:2nd edition, MIT Press, pp. 923-931, 2001.
18) MySQL-Official Site, http://www.mysql.com/
19) SQLite Home Page, https://sqlite.org/
20) OKhttp Home Page : https://github.com/square/okhttp/