Vol.2012-OS-122 No.8
2012/8/1

動的なレプリカ数調節によるデータスパイク平準化

加藤 純[1,a)]　前田 宗則[1,b)]　小沢 年弘[1,c)]

概要：Twitter や YouTube，Yahoo! Video などの大量データを扱うデータインテンシブな大規模サイトで人気データにユーザーのアクセスが集中すること (データスパイク) によるレスポンス性能の低下が問題になっている．本稿では時間軸に沿って重み付けされた人気度を省メモリで推定することによってデータスパイクの検出を高速に行い，データスパイク時のアクセス頻度の高さによって動的にレプリカ数を調節することでデータスパイクによるレスポンス性能の低下問題を解決する ARD(Adaptive Replication Degree) 機構を提案する．ARD 機構の評価を Michael Jackson の突然死に伴って Wikipedia で発生したデータスパイクを模したワークロードで行ったところ，データスパイクが発生した 5 秒後にはそのスパイクを検出でき，データスパイク時の平均レスポンス時間を 70%削減できることを確認した．

Adaptive Replication Degree for Leveling Data Spike

Jun Kato[1,a)]　Munenori Maeda[1,b)]　Toshihiro Ozawa[1,c)]

Abstract: In Internet-scale data-intensive sites (e.g., Twitter, YouTube and Yahoo! Video), extremely concentrated accesses for a piece of popular data, called data spike, cause a high latency problem. This report proposes ARD (Adaptive Replication Degree) to solve such problem. The ARD estimates a time-weighted popularity of data by a memory-efficient manner and quickly detects a data spike through the estimated weighted popularity. When the ARD detects the data spike, it automatically changes the number of replicas of data by access rate of the data spike and levels the concentrated accesses. The evaluation of the ARD is performed by simulating the data spike caused by the sudden death of Michael Jackson at Wikipedia. This result shows that the ARD detects the data spike in five seconds after the rise of the spike and reduces average response time during the data spike by 70%.

1. はじめに

Twitter や Facebook に代表されるソーシャルネットワークサービスやモバイル機器に搭載されたセンサーなどが生み出す大量のデータ (ビッグデータ) を活用することでビジネスの競争力強化やユーザーエクスペリエンスの向上を図る取り組みが広まっている．データ容量が膨大で見積もりも難しいビッグデータの蓄積・活用には優れたスケーラビリティーとコスト削減のためのハードウェアのコモディティ化が必要とされており，それらを実現する新しい分散ストレージシステムとして Amazon Dynamo[1] や Apache Cassandra[2]，Windows Azure Storage[3] などが提案されている．これらのシステムではシステムにサーバを追加・削除するだけで必要に応じた性能・容量の拡張ができるため柔軟性に富んでおり，またデータの複製 (レプリケーション) により高価な RAID 装置を用いなくても安価なコモディティディスクだけで高可用性，高信頼性を実現することができる．

しかし，そのようなサーバの追加・削除による性能拡張では解決できないレスポンス性能の低下問題が Twitter や YouTube，Yahoo! Video など実際に大量データを扱うデータインテンシブな大規模サイトで報告されている [4], [5], [6]．つまり，ある特定のデータにユーザーのアクセスが集中するようなワークロード (**データスパイク**[4]) 下でその人気データを持つサーバのレスポンス性能が低下してしまうことである．例えば 2009 年に Michael Jackson

[1] (株) 富士通研究所
Fujitsu Laboratories Ltd.
[a)] jun.kato@jp.fujitsu.com
[b)] maeda.munenori@jp.fujitsu.com
[c)] t.ozawa@jp.fujitsu.com

が突然死した際，あまりのセンセーショナルさから CBS や CNN，MSNBC などのニュースサイトで軒並み性能低下が観測され [7]，Google ではシステムへの攻撃と誤検出するほど Michael Jackson 関連の検索が急上昇 [8]，Yahoo! では”Michael Jackson rushed to hospital” の記事に 10 分間で 800,000 回ものアクセス [9] が行われた．そのような場合にサーバを追加しても人気データを持つサーバにのみアクセスが集中する状況に変化はなく，人気データを持つサーバのレスポンス性能は低下したままである．

このような問題を解決するためには，データの個数に依存することなく省メモリで人気データを検出する必要がある．人気データはそのデータへのアクセス回数を全データの合計アクセス回数で割った人気度を指標として検出することができるが，この手法ではデータごとにアクセス回数を記録しておく必要があるためにデータの個数に比例して消費メモリ量が増えてしまい，上述のような大量データを扱う分散ストレージシステムに適用することができない．この問題を解決するために人気度の誤差を許容することでメモリ消費量の削減を行うアルゴリズムがいくつか提案 [10], [11], [12] されており，Space Saving アルゴリズムはその中でも特に高速・省メモリ・高精度であることが知られている [13].

しかし，この Space Saving アルゴリズムは時間軸に沿った人気度の重み付けを行っていないため高速にデータスパイクを検出することができない．なぜなら，Space Saving アルゴリズムが算出する人気度は動作開始時点から現在までの平均の人気度であるため，データスパイクが動作開始後から十分長い時間がたって発生した場合に過去の人気度に引きずられて人気度が敏感に変化せず，データスパイクの検出が遅れてしまうからである．そのような突発的なデータスパイクでも高速に検出できるようになるためには時間軸に沿って重み付けされた人気度を推定するアルゴリズムが必要である．

データスパイクはレプリカ数を増やしアクセスを分散させることで平準化を行うことができるが，そのためにはデータスパイク時のアクセス頻度の高さ (データスパイクの強度) に合わせて動的にレプリカ数を調節する必要がある．強度が小さい緩やかなデータスパイクの時に必要以上にレプリカを作ってしまうとシステムリソースを無駄に消費してしまい，強度が強い突発的なデータスパイク時に必要な数だけレプリカを作らないとアクセス集中が解消されず，レスポンスの性能低下問題が残ることになる．これは，データスパイクが過ぎて不必要となった追加レプリカを削除する場合にも当てはまる．削除するレプリカ数が少なすぎるとシステムリソースを無駄に消費したままであり，必要以上に追加したレプリカを削除してしまうと逆にデータスパイクを誘発してしまい，レスポンスの性能低下問題を引き起こすことになる．このようなことを防ぐためにはデータスパイクの強度に合わせたレプリカ数を算出する必要がある．

本稿では，突発的なデータスパイクでも高速に検出することができ，動的に適切な数のレプリカ数を追加・削除することでデータスパイクによるレスポンス性能低下問題を解決する **Adaptive Replication Degree (ARD)** 機構を提案する．ARD 機構は Space Saving アルゴリズム [14] を拡張した **Space Saving for Data Spike (SSDS)** アルゴリズムを人気度推定エンジンとして用いる．SSDS アルゴリズムは Space Saving アルゴリズムの拡張であるためメモリ消費量がデータの個数に依存しておらず省メモリであり，Space Saving が推定していた動作開始時点から現在までの平均の人気度ではなく過去の人気度ほど重みが減少するように重み付けされた人気度を推定しているため，突発的なデータスパイクでも高速に検出することができる．ARD 機構にはレプリカの追加と削除を判定するデータスパイク検出閾値と下限閾値の他にレプリカの追加と削除の予兆を表すデータスパイク予兆閾値と下限予兆閾値があり，この 2 つの予兆閾値と人気度推定エンジンの直近一定時間分の履歴を用いることでデータスパイクの強度に合わせたレプリカ数の調節を行うことができる．

ARD 機構の評価は Michael Jackson 死去に伴って Wikipedia で発生したデータスパイクを参考にして行った．Wikipedia Page Counts[15] で公開されているアクセスログから分かるページの人気度のうち上位 2 割のページのみで評価を行ったところ，ARD 機構を用いることでデータスパイクが発生した 5 秒後には最初のデータスパイクの検出ができ，発生後 20 秒以降では全てのサーバにアクセスが分散され，平均レスポンス時間が 70%削減したこを確認した．

ARD 機構を用いることでシステム運用の簡素化とユーザーエクスペリエンスの向上が見込める．従来ではデータスパイクを回避するために人手によってシステムの性能監視を行い，データスパイクが検出されると手動でデータの再配置を行うことでアクセスの分散を行っていた．ARD 機構を用いることでこれらの処理を自動化し，システム運用を簡素化することができる．SSDS アルゴリズムを使うことにより突発的なデータスパイクでも迅速に対応することができ，データスパイクの強度から必要なレプリカ数を追加することで自動的にデータスパイクを平準化することができる．それにより，ユーザーから見ると従来では人気がありアクセスしにくかったデータに人気に左右されることなく安定してアクセスできるようになるので優れたユーザーエクスペリエンスを提供することができる．

2. ARD 機構の構成

ARD 機構は各データの重み付けされた人気度 P^W を省

メモリで高精度に推定する人気度推定エンジンと，人気度推定エンジンから得られる人気度 P^W とその履歴からレプリカ数の調節を行うレプリカマネージャーの 2 つから構成される．ARD 機構はデータへのアクセスがあるたびに人気度推定エンジンによって人気度 P^W の更新を行い，人気度 P^W がスパイク検出閾値 P_T を超えたものをデータスパイクとして検出する．データスパイクが検出されると，レプリカマネージャーは人気度推定エンジンの履歴からアクセス頻度の算出を行い，その値をもとにデータスパイクを平準化するために必要な追加レプリカ数 k_C を算出する．追加するレプリカ数 k_C が決まると，まだレプリカを持っていないサーバの中から最近の平均アクセス処理数が少ない順に k_C 台を選び，それらのサーバに新しくレプリカを配置する．追加したレプリカの人気度 P^W は定期的に監視され，すべてのレプリカで人気度 P^W が下限閾値 $P_U = \frac{r_I - 1}{r_I} P_T$(ただし，$r_I$ は初期レプリカ数) を下回った場合にレプリカの削除を行う．レプリカの削除を行う場合，レプリカマネージャーは削除できるレプリカ数 k_D をレプリカ追加の時と同様にアクセス頻度から算出する．削除するレプリカ数 k_D が決まると最近レプリカを追加されたサーバから順に k_D 台を選び，それらのサーバが持っているレプリカを削除する．

以下では，ARD 機構を構成する人気度推定エンジンとレプリカマネージャーについて述べる．なお，本稿では実際にレプリカの追加と削除を行う手法については述べておらず既知のものとする．また，重み付けした人気度も単に人気度と表記するが，P^W のように上付き添え字として W をつけることで重み付けされた人気度であることを示す．

2.1 人気度推定エンジン

人気度推定エンジンは人気度 P を最大誤差 ϵ で推定する Space Saving アルゴリズム [14] を拡張した Space Saving for Data Spike (SSDS) アルゴリズムを用いて時間軸に沿って重み付けされた人気度 P^W を推定する．以下では，まず人気度推定エンジンの基盤となる Space Saving アルゴリズムの概略について述べ，次にその拡張である SSDS アルゴリズムについて述べる．

2.1.1 Space Saving アルゴリズムの概略

Space Saving アルゴリズムは図 1 に示す Stream-Summary データ構造を用いて人気度 P の推定を行う．Stream-Summary はデータ名とカウントからなる最大で $\frac{1}{\epsilon}$ 個の要素とそれを管理するバケットからなるデータ構造である．各バケットはカウントが同じ要素をリスト構造で管理しており，バケットは管理している要素のカウント値で昇順にソートされたソート済みリストによって管理される．カウントはデータへのアクセスがあるたびにインクリメントされ，データ D の人気度 P はデータ D のカウント C とカウントの合計値 $N(=\sum_i C_i)$ を用いて $P \simeq \frac{C}{N}$ で推定される．

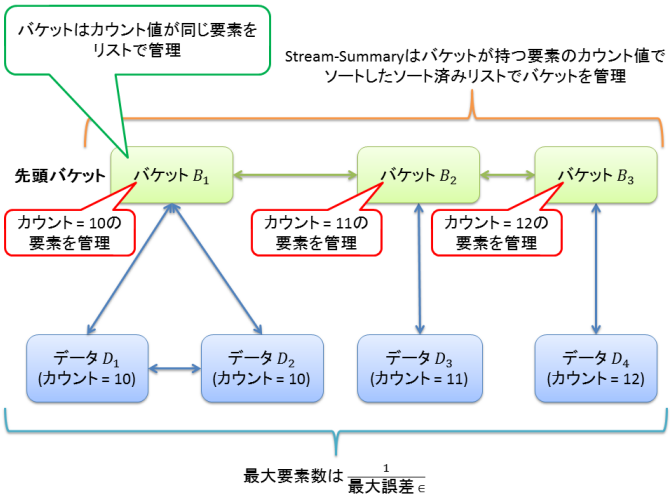

図 1 Stream-Summary データ構造

Require: 人気度の最大誤差 ϵ
function UPDATECOUNTER(データ D) ▷ データへのアクセスごとに呼び出す
if データ D が Stream-Summary に要素として含まれていた **then**
データ D のカウント $+=1$
データ D を管理するバケットを必要なら変更
else if Stream-Summary に含まれる要素数が $\frac{1}{\epsilon}$ より小さい **then**
データ D をカウント $=1$ として Stream-Summary に追加
else ▷ カウントが最小の要素とデータ D の入れ替えを行う
$minCount$ $=$ 先頭バケットが管理しているリストの先頭要素 E のカウント
要素 E の削除
データ D をカウント $=$ $minCount+1$ として Stream-Summary に追加
end if
end function

図 2 データアクセス時のカウント更新アルゴリズム

データ D へのアクセスがあった際，Space Saving は図 2 に示すアルゴリズムを用いてカウントの更新を行う．データ D が Stream-Summary に要素として含まれていた場合，その要素のカウントをインクリメントする．それによりデータ D を管理するバケットが変更される際はデータ D を管理するバケットの変更を行う．データ D が Stream-Summary に含まれていない場合，Stream-Summary の要素数に空きがあるかを調べる．要素数が $\frac{1}{\epsilon}$ より小さい場合は最大要素数まで達していないのでデータ D をカウント $=1$ として Stream-Summary に追加する．最大要素数まで達していて要素数に空きがない場合，先頭バケットが管理しているリストの先頭要素 (カウントを $minCount$ とする) を削除する代わりにデータ D をカウント $=$ $minCount+1$ として Stream-Summary に追加することでカウントが最小の要素とデータ D の入れ替えを行う．

Require: カウントの合計値 N の閾値 N_T
Require: 重み係数 $\alpha(\frac{r_I}{r_I+1} < \alpha < 1)$
function SHRINKALLCOUNTERS ▷ N が N_T に達するたびに呼び出す
$totalCount = 0$
for all バケット B **do**
for all 要素 E in バケット B **do**
E のカウントを $(1-\alpha)$ 倍 (小数点以下切り上げ) ▷ カウントの更新
$totalCount+ = E$ のカウント
end for
end for
for all バケット B **do**
while B に隣接するバケットが同じカウントの要素を管理している **do**
B と隣接バケットをマージ ▷ カウントの丸め誤差による不整合の解決
end while
end for
$N = totalCount$ ▷ カウントの合計値 N の更新
end function

図 3 N が N_T に達した時のカウント更新アルゴリズム

このようにカウントの更新を行うことで人気度 P の推定誤差は $0 \leq \frac{C}{N} - P < \epsilon$ となる．これは以下から導かれる．データ D の推定人気度に誤差が生じるのは，データ D のカウントがデータ D が要素の入れ替えによって Stream-Summary 構造体を追い出されてから再び戻るまでの間にデータ D 以外の要素へのアクセスによって増加するためである．これによりカウントが本来の値より大きい値となるので推定人気度は常に人気度 P よりも高く $0 \leq \frac{C}{N} - P$ である．このカウントの差が最大になるのは $minCount = C - 1$ がすべてデータ D 以外の要素へのアクセスだった場合，つまり初めてデータ D にアクセスが行われたときでその差は $minCount = C - 1$ となる．$N = N - 1$ 時点の最小カウントである $minCount$ が最大になるのは $\frac{1}{\epsilon}$ 個の全てのカウントが等しいときなのでカウントの差は最大で $C - 1 = \epsilon(N - 1)$ である．したがって，$\frac{C}{N} - P < \epsilon - \frac{\epsilon}{N}$ となるので $\frac{C}{N} - P < \epsilon$ である．

2.1.2 SSDS アルゴリズム

SSDS アルゴリズムは 2.1.1 で述べた Space Saving アルゴリズムに図 3 に示すカウント更新アルゴリズム (カウントシュリンク) を追加したものである．重み係数 $\alpha(\frac{r_I}{r_I+1} < \alpha < 1)$ は人気度に重み付けを行う定数で 1 に近いほど最近の人気度を重視する．SSDS アルゴリズムはカウントの合計 N が閾値 N_T に達したとき，このカウント更新アルゴリズムを呼び出し全てのカウントを $1-\alpha$ 倍する．それによりカウントが整数値でなくなった場合はカウントを整数化するために小数点以下を切り上げる．このカウントの丸め誤差により隣接バケットが同じカウントの要素を持つ場合は Stream-Summary の整合性を保つためにそれらバケットのマージを行う．カウントを $1-\alpha$ 倍したことによりカウントの合計値 N も更新されるが，カウントの丸め誤差により更新後の N は $(1-\alpha)N_T$ より最大で Stream-Summary の最大要素数 $(= \frac{1}{\epsilon})$ だけ大きい値になる．SSDS アルゴリズムはこのような処理を閾値 N_T に達するたびに行う．以下では，どの時点を述べているかを示すために n 回目にカウントの合計 N が閾値 N_T に達してカウントシュリンクが終了した直後から $n+1$ 回目に閾値 N_T に達してカウントシュリンクを行うまでの期間を n 番目のフェイズと呼ぶことにする．

n 番目のフェイズで SSDS アルゴリズムが算出する推定人気度 $\frac{C}{N}$ は次のように重み付けされた人気度 P^W を推定する．

$$P^W = (1-\beta)P_{n-1}^{EMA} + \beta P_n \quad (1)$$
$$(\text{ただし}, P_{n-1}^{EMA} = \alpha P_{n-1} + (1-\alpha)P_{n-2}^{EMA})$$

ここで，$\beta(0 \leq \beta \leq \alpha)$ はそのフェイズの進行具合を示しており，フェイズ開始時点のカウントの合計を N_I とし，そのフェイズ中にアクセスが M 回あったとすると $\beta = \frac{M}{N_I + M}$ で与えられる値である．P_n^{EMA} は平滑化係数を α とした今までのフェイズの人気度 P の指数移動平均 (Exponential Moving Average) であり，$n-1$ 番目のフェイズの人気度を P_{n-1} とすると $\alpha P_{n-1} + (1-\alpha)P_{n-2}^{EMA}$ で与えられるように α で重み付けされた人気度である．P_n はこのフェイズ中が始まってから現在に至るまでの期間の人気度を示しており，P^W はフェイズ開始直後には過去の人気度にあたる P_{n-1}^{EMA} を重視するが，フェイズが進むごとにこのフェイズの人気度 P_n を重視するような人気度である．以下では推定人気度 $\frac{C}{N}$ と人気度 P^W の誤差を求め，レプリカの追加・削除を行う上でその誤差を無視して $P^W \simeq \frac{C}{N}$ とみなすためには $\epsilon, \alpha, N_T, P_T$ をどのように決める必要があるかを述べる．

フェイズ開始時点 $(\beta = 0)$ における $P^W = P_{n-1}^{EMA}$ と SSDS アルゴリズムが算出する推定人気度 $\frac{C}{N}$ の誤差は

$$-P_T\epsilon^- - \frac{P_T}{\alpha^2\epsilon N_T} < \frac{C}{N} - P^W < \epsilon^+ + \frac{1}{\alpha}(\epsilon + \frac{1}{N_T}) \quad (2)$$

になる．ここで，ϵ^-, ϵ^+ は $f(x) = \frac{x}{(1-\alpha)N_T + x}$ として $\epsilon^- = f(\frac{1}{\epsilon} - 1), \epsilon^+ = f(1)$ で与えられる値である．この証明は次の通りである．

証明. まずは，カウントシュリンクによる推定人気度 $\frac{C}{N}$ の変動が $-P_T\epsilon^-$ 以上 ϵ^+ 以下であることを示す．n 番目のフェイズが終了した時点でのカウントを C_n，カウントシュリンクにより生じるカウントとその合計値の丸め誤差をそれぞれ $C_\varepsilon(0 \leq C_\varepsilon < 1), N_\varepsilon(0 \leq N_\varepsilon < \frac{1}{\epsilon})$ とすると，カウントシュリンクによる変動は $C_\varepsilon = N_\varepsilon = 1$ の時が最大，$C_\varepsilon = 0, N_\varepsilon = \frac{1}{\epsilon} - 1$ の時が最小であり，ARD 機構は

後述のように推定人気度 $\frac{C}{N}$ が P_T を超えないようにレプリカマネージャーが調節を行うので $0 \leq \frac{C_n}{N_T} < P_T$ から

$$-P_T\epsilon^- < \frac{(1-\alpha)C_n + C_\varepsilon}{(1-\alpha)N_T + N_\varepsilon} - \frac{C_n}{N_T} < \epsilon^+$$

となるので，カウントシュリンクによる変動は $-P_T\epsilon^-$ 以上 ϵ^+ 以下である．したがって，n 番目のフェイズが終了した時点で

$$-\frac{P_T}{\alpha^2\epsilon N_T} < \frac{C_n}{N} - P_n^{EMA} < \frac{1}{\alpha}(\epsilon + \frac{1}{N_T}) \qquad (3)$$

となっていることを示す．この証明にはフェイズについての数学的帰納法を用いる．初回のフェイズについては Space Saving アルゴリズムより (3) が満たされるので，n 番目のフェイズ終了時点で (3) が成り立つことを仮定して $n+1$ 番目のフェイズ終了時点でも (3) が成り立つことを示す．$n+1$ 番目のフェイズは，開始時点ではカウントが $(1-\alpha)C_n + C_\varepsilon$ であり，終了時点までに $P_{n+1}(\alpha N_T - N_\varepsilon)$ 回アクセスが行われた場合の人気度を Space Saving アルゴリズムにより処理を行っているので

$$0 \leq \frac{C_{n+1}}{N_T} - \frac{(1-\alpha)C_n + C_\varepsilon + P_{n+1}(\alpha N_T - N_\varepsilon)}{N_T} < \epsilon \qquad (4)$$

となる． $C_\varepsilon - P_{n+1}N_\varepsilon$ は推定人気度が P_T を超えないことから $(1-\alpha)C_n + C_\varepsilon + P_{n+1}(\alpha N_T - N_\varepsilon)P_{n+1} < N_T P_T$，つまり $0 \leq P_{n+1} < \frac{P_T}{\alpha}$ であることを用いると $-\frac{P_T}{\alpha\epsilon}(C_\varepsilon = 0, N_\varepsilon = \frac{1}{\epsilon}, P_{n+1} = \frac{P_T}{\alpha})$ が下限，$1(C_\varepsilon = 0, P_{n+1} = 0)$ が最大となるので

$$-\frac{P_T}{\alpha\epsilon N_T} < \frac{C_{n+1}}{N_T} - \left\{(1-\alpha)\frac{C_n}{N_T} + \alpha P_{n+1}\right\} < \epsilon + \frac{1}{N_T}$$

帰納法の仮定を用いてまとめると

$$-\frac{P_T}{\alpha^2\epsilon N_T} < \frac{C_{n+1}}{N_T} - P_{n+1}^{EMA} < \frac{1}{\alpha}(\epsilon + \frac{1}{N_T})$$

となり，$n+1$ 番目のフェイズ終了時点でも (3) が成り立つ． □

フェイズ中 $(\beta \neq 0)$ における P^W と SSDS アルゴリズムが算出する推定人気度 $\frac{C}{N}$ の誤差は，(2) のフェイズ開始時点の誤差が十分小さいとき 0 以上 ϵ 未満となる．フェイズ開始時点のカウントとその合計をそれぞれ C_I, N_I，そのフェイズ中に現在に至るまでアクセスが M 回あったとして，(4) の導出と同様に考えると

$$0 \leq \frac{C}{N} - \frac{C_I + P_n M}{N_I + M} < \epsilon$$

である．(2) のフェイズ開始時点の誤差が十分小さいとき，$C_I \simeq P_{N-1}^{EMA} N_I$ が成り立つので

$$0 \leq \frac{C}{N} - \left\{(1-\beta)P_{N-1}^{EMA} + \beta P_n\right\} < \epsilon \qquad (5)$$

となり，人気度 P^W と推定人気度との誤差は 0 以上 ϵ 未満となる．

以上から，ARD 機構が推定人気度 $\frac{C}{N}$ を人気度 P^W とみなしてデータスパイクと不要な追加レプリカを検出するためには (2) の誤差を十分小さくすればよいことが分かる．(2) のフェイズ開始時点の誤差は (5) のフェイズ中の誤差よりも大きいので，(2) の誤差を十分小さくすれば (5) の誤差も十分小さくなる．したがって，(2) の誤差が下限閾値 P_U に比べて十分小さくなるように $\epsilon, \alpha, N_T, P_T$ を決めればスパイク検出閾値 P_T と比べても誤差は十分小さくなり，ARD 機構が推定人気度 $\frac{C}{N}$ を使いデータスパイクや不要な追加レプリカを検出する際の誤差を無視することができる．以下では，簡単のために $\epsilon, \alpha, N_T, P_T$ を (2) の誤差が下限閾値 P_U に比べて十分小さくなるようにうまく決めれたとして，人気度推定エンジンが算出する値は人気度 P^W であると仮定する．

2.2 レプリカマネージャー

レプリカマネージャーは人気度 P^W がデータスパイク検出閾値 P_T を超えたデータをデータスパイクとして検出する．人気度 P^W が閾値 P_T を超える場合は P^W の定義 (1) から今までのフェイズの人気度の指数移動平均 P_{n-1}^{EMA} とこのフェイズの人気度 P_n が閾値 P_T に近い値になっている場合と，指数移動平均 P_{n-1}^{EMA} は閾値 P_T に比べて十分小さいがこのフェイズの人気度 P_n が閾値 P_T に比べて十分大きい場合の 2 通りが考えられる．これらはそれぞれ人気度 P が緩やかに上がる場合と突発的に跳ね上がる場合に当たり，レプリカマネージャーは人気度 P^W の一指標だけで人気度 P の上がり方に関わりなくデータスパイクを検出することができる．以下では，レプリカの追加・削除を検出した後にアクセス頻度から追加・削除するレプリカ数を算出する手法について述べ，次にレプリカを追加・削除する際に必要な人気度推定エンジンの更新処理について述べる．

2.2.1 レプリカ数の算出

レプリカマネージャーは図 4 に示すように，データスパイク時には人気度 P^W がデータスパイク予兆閾値 $P_T^C = \alpha P_T$ を超えてからスパイク検出閾値 P_T を超えた現在までの期間の，不要な追加レプリカを削除する場合は人気度 P^W が下限予兆閾値 $P_U^C = \frac{r_I - \alpha}{r_I} P_T^W$ を下回ってから下限閾値 P_U を下回るまでの期間のアクセス頻度を人気度推定エンジンの直近一定時間分の履歴から求め，そのアクセス頻度をもとに追加・削除に必要なレプリカ数を算出する．そこで，まずはアクセス頻度の算出手法について述べ，次にアクセス頻度からレプリカ数を算出する手法について述べる．

レプリカマネージャーはアクセス頻度を求めるために定期的なタイミングとカウントシュリンクが行われる直前と

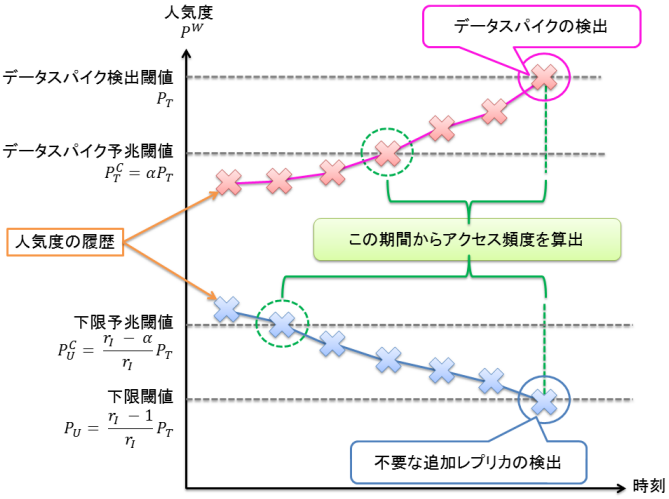

図 4 アクセス頻度の算出に用いる期間

直後のタイミングで次の 3 つを履歴に記載する．

(i) 人気度 P^W がデータスパイク予兆閾値 P_T^C を超えているレプリカのカウント

(ii) レプリカ数が r_I より大きいレプリカのカウント

(iii) 前回の履歴記録から今回の履歴記録までの間に人気度 P^W が予兆閾値 P_T^C を超えたレプリカのデータ名と予兆閾値 P_T^C を超えた時刻のタプルリスト

(i) と (ii) は履歴記録タイミング時点でのカウントを人気度推定エンジンから取得し履歴に記載する．(iii) のリストは人気度推定エンジンが人気度 P^W の更新を行うたびにその更新によって人気度 P^W がデータスパイク予兆閾値 P_T^C を上回るレプリカがあるか監視を行うことで取得できる．人気度 P^W が予兆閾値 P_T^C を超えていた場合はその時の時刻とデータ名を記録しておき，次回の履歴記録タイミング時に (iii) のリストに記載する．

人気度 P^W がデータスパイク予兆閾値 P_T^C を超えてからスパイク検出閾値 P_T を超えるまでの期間のアクセス頻度は履歴の (i) と (iii) から次のように求める．レプリカマネージャーは新しい履歴から順々に直近一定時間分の履歴を辿っていき，追加するレプリカが (iii) のリストに含まれている履歴を探す．そのような履歴が見つかればその履歴から，見つからなければ最後に検索した履歴から人気度 P^W が閾値 P_T を超えるまでの間のアクセス数 Q とその間の時刻の差 t からアクセス頻度 A は $A = \frac{Q}{t}$ として求められる．予兆閾値 P_T^C を超えている間のアクセス数 Q はフェイズごとにアクセス数 Q_n を求め，それらの総和を取ることで求めることができる．フェイズの開始と終了にあたるカウントシュリンクの直後と直前が履歴に含まれているためそのフェイズ中で予兆閾値 P_T^C を超えている時の最古と最新の履歴を取得することができ，それらの履歴の (i) に含まれるカウントの差がそのフェイズ中で予兆閾値 P_T^C を超えている時のアクセス数 Q_n である．

人気度 P^W が予兆閾値 P_T^C を超えた後の履歴を用いることで，突発的なデータスパイク時に正確にデータスパイク中のアクセス頻度を算出することができる．人気度 P^W が予兆閾値 $P_T^C = \alpha P_T$ を超えている間は P^W の定義 (1) から

$$P_n \geq \frac{1}{\beta}(\alpha P_T - (1-\beta)P_{n-1}^{EMA}) \geq P_T - \frac{1-\beta}{\alpha}P_{n-1}^{EMA}$$

であり，突発的なデータスパイクが発生した場合は過去の人気度にあたる P_{N-1}^{EMA} を $P_{N-1}^{EMA} \simeq 0$ と近似できるので $P_n \geq P_T$ となる．これは現在のフェイズの人気度 P_n がデータスパイク検出閾値 P_T を超えているのですでにデータスパイクが発生しているが，重み付けされた人気度 P^W からはまだデータスパイクが検出されていないという状況に当たる．したがって，予兆閾値 P_T^C を超えた後のアクセス頻度はデータスパイクが発生していない時のアクセス頻度を含まないため，データスパイク中のアクセス頻度を正確に表していることになる．

人気度 P^W が下限予兆閾値 P_U^C を下回ってから下限閾値 P_U を下回るまでのアクセス頻度は履歴の (ii) から次のように求める．レプリカマネージャーは新しい履歴から順々に (ii) を見ながら下限予兆閾値 P_U^C を上回る履歴が見つかるまで直近一定時間分の履歴を辿る．そのような履歴が見つかればその履歴から，1 つも見つからなければ 2 番目に新しい履歴から，全てを辿ったが見つからなかった場合は最後の履歴から人気度 P^W が下限閾値 P_U を下回るまでのアクセス頻度を，レプリカ追加時にアクセス頻度を求めた手法と同様にフェイズごとのアクセス数 Q_n から求める．

人気度 P^W が下限予兆閾値 P_U^C を下回った後の履歴を用いることで，現在の人気度が下限閾値 P_U を下回らないような期間がアクセス頻度の算出には含まれなくなる．人気度 P^W が下限予兆閾値 P_U^C を下回っている間は P^W の定義 (1) から $P^W < (1 - \frac{\beta}{r_I}P_T)$ を満たす．これはこのフェイズの人気度 P_n が下限閾値 P_U を下回るための必要条件となっており，実際に $P_n < P_U$ とすると

$$P^W < (1-\beta)P_{n-1}^{EMA} + \beta P_U < (1-\beta)P^T + \beta P_U = (1-\frac{\beta}{r_I})P_T$$

である．したがって，現在のフェイズの人気度 P_n が下限閾値 P_U を下回らないような期間はアクセス頻度の算出には含まれなくなるため，アクセス頻度を不要に高く見積もることがなくなる．

アクセス頻度 A が求まると，レプリカマネージャーはアクセス頻度が閾値 A_T を上回らないように追加・削除するレプリカ数 k_C, k_D を求める．ここで閾値 A_T はサーバが 1 秒間に処理できる最大アクセス数 A_{max} から $A_T = P_T A_{max}$ で求められる値であり，1 つのデータに対しての最大アクセス頻度に当たる値である．レプリカマネージャーはデー

タスパイクを検出した場合 $k_C = \max(1, \lceil \frac{A - A_T}{A_T} r \rceil)$ 個のレプリカを追加する．レプリカ数が r から $r + k_C$ に変わるのでアクセス頻度は $\frac{r}{r + k_C}$ 倍に減り，常に閾値 A_T 以下に保つことができるのでアクセス集中によるレスポンス性能の低下問題を防ぐことができる．追加するレプリカ数 k_C はアクセス頻度が高いほど大きくなるため，データスパイクの強度が強い突発的なデータスパイクの場合にその強さに応じた数のレプリカを追加することができる．また，そのような時にはスパイク予兆閾値 P_T^C 以降のアクセス頻度は正確にデータスパイク中の頻度を表しているのでレプリカの数は適切に算出されていることになる．

レプリカを削除する場合は $k_D = \lfloor \frac{A_T - A}{A_T} r \rfloor$ 個のレプリカを削除する．$A \leq P_U A_T = \frac{r_I - 1}{r_I} P_T A_T$ から

$$\frac{A_T - A}{A_T} r \geq (1 - \frac{A}{A_T} \frac{r_I - 1}{r_I}) r \geq (1 - \frac{r_I - 1}{r_I}) r = \frac{r}{r_I} \geq 1$$

となり，人気度 P^W が下限閾値 P_U を下回った場合にはレプリカを 1 つ以上削除することができる．レプリカ数が r から $r - k_D$ に変わるので削除しないレプリカでのアクセス頻度 A は $\frac{r}{r - k_D}$ 倍に増えるが，$P^W \leq P_U$ の場合は常に $\frac{r}{r - k_D} A \leq A_T$ であるためレプリカを削除することでアクセス頻度が閾値 A_T を上回りデータスパイクを誘発することがない．また，人気度 P^W が下限予兆閾値 P_U^C を下回った後のアクセス頻度を用いることでアクセス頻度を不要に高く見積もることがなくなり，それによって不要なレプリカ数を少なく見積もることがなくなるのでシステムリソースの無駄な消費を削減することができる．

2.2.2 レプリカ追加・削除時の更新処理

新しくレプリカの追加が行われたサーバでは，そのレプリカの人気度 P^W が基準人気度 $P_B = \frac{P_T^C + P_U^C}{2}$ に達するまでアクセスがあったとみなして人気度推定エンジンの更新を行う．基準人気度 P_B はスパイク予兆閾値 P_T^C と下限予兆閾値 P_U^C の平均値であるため，レプリカ追加直後に再びレプリカ追加・削除するような二重操作を防ぐことができる．また，もともとレプリカを持っていたサーバでも二重操作を防ぐために追加を行ったレプリカの人気度 P^W が $\min(P^W, P_B)$ となるように人気度推定エンジンの更新を行い，そのレプリカに関するすべての履歴を削除する．

レプリカが削除されたサーバでは，人気度推定エンジンに削除したレプリカが含まれていた場合はそのカウント分だけカウントの合計値 N を減らしたのちに削除を行い，そのレプリカに関するすべての履歴を削除する．また削除後もレプリカを持っているサーバでは，レプリカ追加時と同様にレプリカの人気度 P^W が $\max(P^W, P_B)$ となるように人気度推定エンジンの更新とそのレプリカに関するすべての履歴の削除を行うことで二重操作を防ぐ．

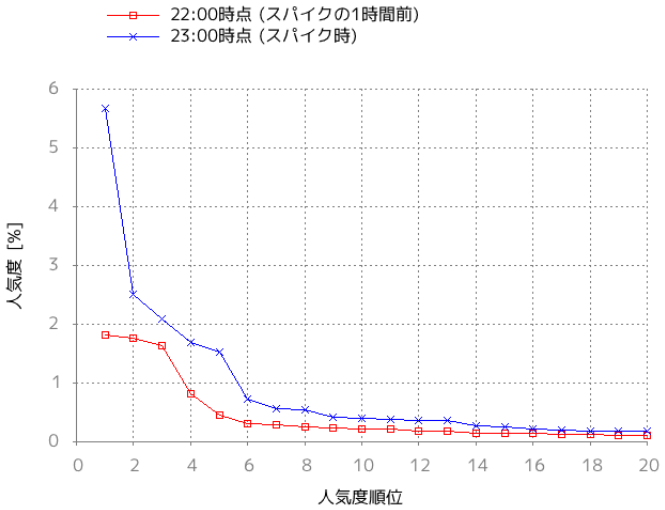

図 5 Michael Jackson スパイク時とその 1 時間前での人気度分布

3. 評価実験

ARD 機構の評価実験は Intel® Xeon® CPU E31270 (3.40GHz,8 コア,HTT 有効), 16GB メモリー, 500GB SATA ディスクで構成されたマシン 31 台が Cisco Catalyst 4948-S ギガビットスイッチで直結されているクラスター上で行った．各マシンでは Linux 3.1.0(Fedora 16) が動作しており，マシン間の通信には TCP(TCP_NODELAY オプション有効) を用いた．マシン 31 台のうち 15 台を負荷かけクライアント，残りの 16 台をサーバとして使用する．サーバははじめ各データに対してレプリカを 3 つ持っており，クライアントがどのサーバにアクセスするかはコンシステントハッシュ法 [16] によって一意に決められる．

ARD 機構の評価実験に使う負荷は Michael Jackson 死去に伴って Wikipedia で発生したデータスパイクを参考に作成した．Wikipedia では 1 時間単位でのページごとのアクセス回数と転送量が Wikipedia Page Counts[15] で公開されているおり，負荷かけクライアントはこのアクセスログから得られる人気度をもとにしてアクセス回数が多い約 20%のページに対してその人気度に応じたアクセスを行うことで Michael Jackson 死去時のデータスパイクを模倣した負荷をかける．実験では Michael Jackson 死去に伴いそのページが急激に人気になる 2009/6/25 23:00 とその 1 時間前の 22:00 のアクセスログを用いた．図 5 は公開アクセスログから得られる 22:00 時点と 23:00 時点の上位 20 位までの人気度分布を示しており，23:00 時点の人気度 1 位が Michael Jackson ページである．この図から 23:00 時点の Michael Jackson ページは 22:00 時点で人気度が 1 位のページよりも非常に人気度が高いことが分かる．そのため，23:00 時点の負荷をかけると Michael Jackson ページを持つサーバにアクセスが集中することになる．実験ではスパイク前の負荷として 22:00 時点の負荷を 5 分間かけて

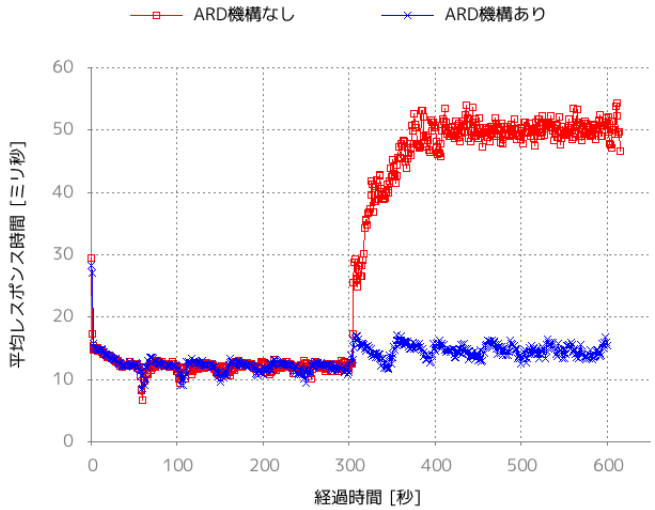

図 6 全クライアントの平均レスポンス時間

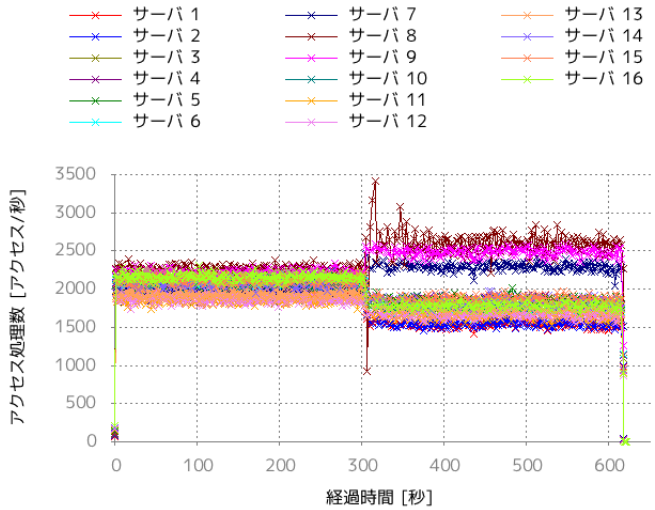

図 7 サーバのアクセス処理数 (ARD 機構なし)

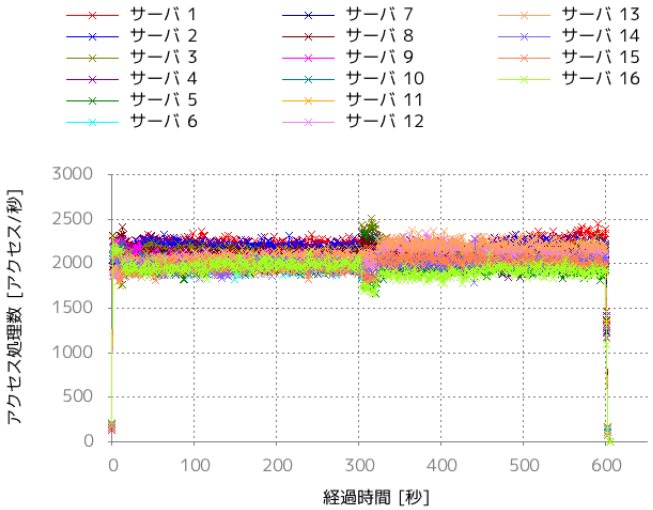

図 8 サーバのアクセス処理数 (ARD 機構あり)

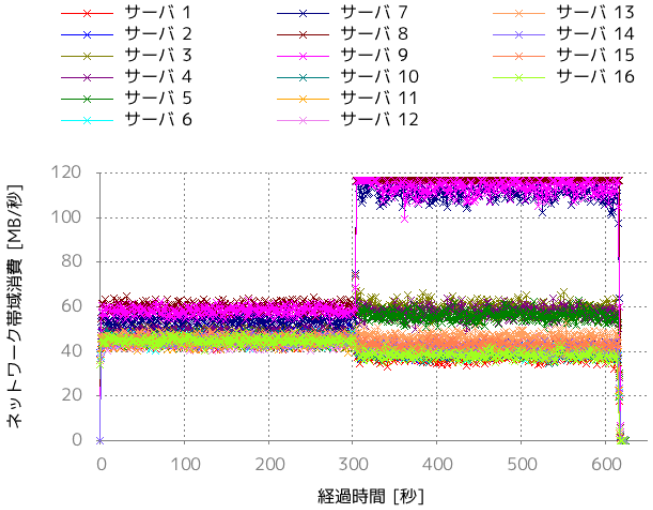

図 9 サーバのネットワーク消費帯域 (ARD 機構なし)

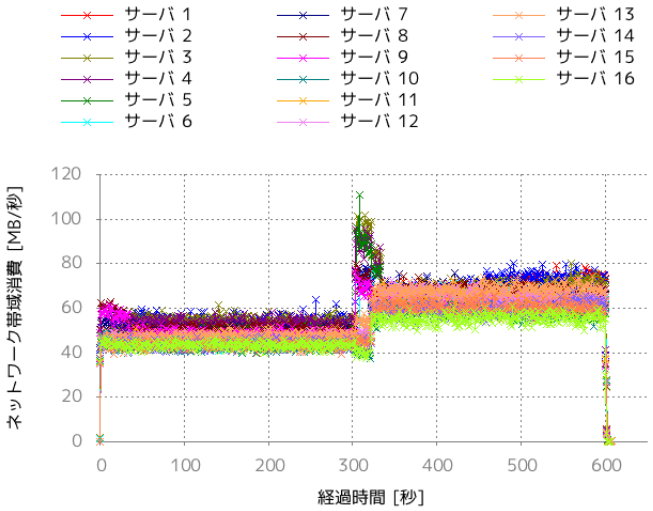

図 10 サーバのネットワーク消費帯域 (ARD 機構あり)

から 23:00 時点の負荷に切り替えを行い，負荷を切り替えた後の負荷かけクライアントからみた平均レスポンス時間で ARD 機構の評価を行う．

実験では，ARD 機構のパラメーターとして $\epsilon = 0.05\%, \alpha = 0.875, N_T = 1048576, P_T = 2.0\%, A_{max} = 3,000$ を用いた．この時，人気度推定エンジンが推定する人気度 P^W のフェイズ開始時点の誤差は (2) から $-0.03\% < \frac{C}{N} - P^W < 0.06\%$ となる．これは下限閾値 $P_U = 1.33\%$ と比べて 2 桁小さい値であるため，人気度推定エンジンの誤差は無視することができる．レプリカマネージャーは 5 秒ごと 60 秒間の履歴を保持しており，その履歴とデータスパイク予兆閾値 $P_T^C = 1.75\%$，下限予兆閾値 $P_U^C = 1.42\%$ を用いてアクセス頻度の算出を行う．

実験結果の図 6 は全クライアントの平均レスポンス時間を示しており，ARD 機構を用いることでデータスパイクを迅速に検出できると同時にレスポンス性能の低下を防いでいることが分かる．ARD 機構がない場合でデータスパイク時の負荷をかけると平均レスポンス時間が 50 ミリ秒にまで増加しているが，これは Michael Jackson ページを

持っているサーバにアクセスが集中するためそのサーバのネットワーク帯域が飽和してしまうからである．図 7 は ARD 機構がない場合の各サーバのアクセス処理数を示しており，Michael Jackson ページを持つサーバ 7，8，9 の 3 サーバのみが他のサーバよりも多くアクセスを処理している．アクセス処理数が多いこの 3 つのサーバは，ARD 機構がない場合のサーバのネットワーク消費帯域を示している図 9 から分かるように 1 ギガビット近くネットワーク帯域を消費しており，平均レスポンス時間を押し上げる要因となっている．これに対し，ARD 機構がある場合では図 8 のアクセス処理数でも図 10 のネットワーク消費帯域でも特定のサーバのみが飛び抜けて過負荷になっているいるということがない．図 6 から分かるように ARD 機構はデータスパイク時の負荷に切り替えた 5 秒後にデータスパイクの検出を行い，レスポンス性能が下がらないようにデータスパイクの平準化を行っている．ARD 機構はその 15 秒後に再びデータスパイクを検出しており，その際にレプリカを 16 個まで増やしている．このことから，データスパイク発生後 20 秒以降では Michael Jackson ページを全てのサーバで処理するようになるので，特定のサーバのみにアクセスが集中してレスポンス性能が下がることがない．実際に，クライアントから見たレスポンス時間はデータスパイクが発生した直後から 15 ミリ秒と安定しており，ARD 機構がなかった場合と比較してデータスパイク時の平均レスポンス時間を 70%削減していることが分かる．

4. 関連研究

データスパイクによる性能劣化問題を解消する研究はいくつかある．Intelligent Workload Factoring 機構 [6] はシステムへの負荷が閾値を超えたら fastTopK アルゴリズムを用いて人気なデータの検出を行い，その人気データをパブリッククラウドに移行することで負荷のオフロードを行う．しかし，この機構は ARD 機構のようにデータの人気度ではなくシステムへの負荷をもとにデータスパイクを検出するため，データスパイクが発生しても負荷が変化しない場合にはデータスパイクの検出ができない．実際，2009 年の Michael Jackson 死去時に Wikipedia で発生したデータスパイクではワークロードに変動がなかったことが報告されており [4]，そのような場合にデータスパイクを検出することができない．

動的にレプリカ数を調節する研究は目的によって次の 2 つ, (1) データの可用性や耐久性の向上, (2) 性能の向上, に大別される．ARD 機構は性能の向上を目的としており，データの可用性や耐久性の向上を目指した研究 [17], [18], [19] とは目的が異なる．

人気データのレプリカ数を動的に調節することで性能を向上させる研究はストレージの分野に限らず P2P やグリッドコンピューティングの分野でも行われてきた．EAD アルゴリズム [20] は P2P システム上でクエリレートの指数移動平均をもとに人気データの検出を行い，各クエリが共通して最も通るノードに新しいレプリカを配置することでクエリ経路の短縮とレプリカ利用効率の向上を行う．LALW アルゴリズム [21] は複数のクラスタから構成される階層型データグリッド上でアクセス頻度の指数移動平均をもとに人気データの検出を行い，アクセス頻度が高くなると予測されたクラスタにレプリカを追加することでアクセス遅延とネットワーク帯域消費の低減を行う．これらのアルゴリズムはアクセスがあったデータごとに人気度を算出するためデータの個数が膨大になるとメモリ消費量も比例して大きくなってしまう問題がある．ARD 機構は人気度の算出に最大でも $\frac{1}{\epsilon}$ 個のデータしか用いないためそのようなことはない．

5. おわりに

本稿では，省メモリに重み付けされた人気度を推定することで高速にデータスパイクを検出することができ，データスパイクの強度によって動的にレプリカ数を変えることでデータスパイク時のレスポンス性能低下問題を解決する ARD 機構の提案を行った．ARD 機構の人気度推定エンジンは SSDS アルゴリズムを用いているため，メモリ消費量がデータの個数に依存することがなく省メモリであり，重み付けされた人気度を推定するため突発的なデータスパイクでも高速に検出できる．ARD 機構のレプリカマネージャーは人気度推定エンジンが推定した人気度が，レプリカを追加する場合にはデータスパイク予兆検出閾値を超えてからデータスパイク検出閾値を超えるまでの間の，レプリカを削除する場合は下限予兆閾値を下回ってから下限閾値を下回るまでの間のアクセス頻度を人気度推定エンジンの直近一定時間分の履歴から求め，その値から追加・削除に必要なレプリカ数を算出することでデータスパイクの強度にあわせたレプリカ数の調節を行う．ARD 機構の評価は Wikipedia で発生した Michael Jackson スパイクを模したワークロードで行った．ARD 機構を用いることでデータスパイク発生後 5 秒後にはデータスパイクを検出でき，データスパイク時の平均レスポンス時間を 70%削減できることを確認した．

今後の課題として，データサイズの考慮が挙げられる．現状では重み付けされた人気度をもとにレプリカ数の算出を行っているが，データサイズが異なるとデータへのアクセス時間も異なってくるため，同じ数だけレプリカを追加しても性能が同じように向上するとは限らない．そのため，データスパイク検出閾値をデータサイズによって変えることでアクセス時間がかかるサイズの大きいデータほど早めにデータスパイクとして検出をしたり，データサイズに応じて追加・削除するレプリカ数を変更することが必要であると考えている．

参考文献

[1] DeCandia, G. et al.: Dynamo: Amazon's Highly Available Key-Value Store, *SIGOPS Operating System Review*, Vol. 41, No. 6, pp. 205–220 (2007).
[2] Lakshman, A. and Malik, P.: Cassandra - A Decentralized Structured Storage System, *SIGOPS Operating System Review*, Vol. 44, No. 2, pp. 35–40 (2010).
[3] Calder, B. et al.: Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency, *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 143–157 (2011).
[4] Bodik, P., Fox, A., Franklin, M. J., Jordan, M. I. and Patterson, D. A.: Characterizing, Modeling, and Generating Workload Spikes for Stateful Services, *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 241–252 (2010).
[5] Torres, R., Finamore, A., Kim, J. R., Mellia, M., Munafo, M. M. and Rao, S.: Dissecting Video Server Selection Strategies in the YouTube CDN, *31st International Conference on Distributed Computing Systems (ICDCS)*, pp. 248–257 (2011).
[6] Zhang, H., Jiang, G., Yoshihira, K., Chen, H. and Saxena, A.: Intelligent Workload Factoring for a Hybrid Cloud Computing Model, *World Conference on Services - I*, pp. 701–708 (2009).
[7] Shiels, M.: Web slows after Jackson's death, BBC News (online), available from ⟨http://news.bbc.co.uk/2/hi/8120324.stm⟩ (accessed 2012-06-28).
[8] Pittman, R.: Outpouring of searches for the late Michael Jackson, Google (online), available from ⟨http://googleblog.blogspot.jp/2009/06/outpouring-of-searches-for-late-michael.html⟩ (accessed 2012-06-28).
[9] Dugan, N.: Losing Michael Jackson, Yahoo! (online), available from ⟨http://ycorpblog.com/2009/06/26/losing-michael-jackson/⟩ (accessed 2012-06-28).
[10] Karp, R. M., Shenker, S. and Papadimitriou, C. H.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags, *ACM Transactions on Database Systems (TODS)*, Vol. 28, No. 1, pp. 51–55 (2003).
[11] Cormode, G., Korn, F., Muthukrishnan, S. and Srivastava, D.: Space- and Time-Efficient Deterministic Algorithms for Biased Quantiles over Data Streams, *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 263–272 (2006).
[12] Schweller, R. et al.: Reversible Sketches: Enabling Monitoring and Analysis Over High-Speed Data Streams, *IEEE/ACM Transactions on Networking (TON)*, Vol. 15, No. 5, pp. 1059–1072 (2007).
[13] Cormode, G. and Hadjieleftheriou, M.: Methods for finding frequent items in data streams, *The VLDB Journal*, Vol. 19, No. 1, pp. 3–20 (2010).
[14] Metwally, A., Agrawal, D. and Abbadi, A. E.: An Integrated Efficient Solution for Computing Frequent and Top-k Elements in Data Streams, *ACM Transactions on Database Systems (TODS)*, Vol. 31, No. 3, pp. 1095–1133 (2006).
[15] Wikimedia: Page view statistics for Wikimedia projects, Wikimedia (online), available from ⟨http://dumps.wikimedia.org/other/pagecounts-raw/⟩ (accessed 2012-06-28).
[16] Karger, D. et al.: Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web, *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663 (1997).
[17] Zhong, M., Shen, K. and Seiferas, J.: Replication Degree Customization for High Availability, *SIGOPS Operating System Review*, Vol. 42, No. 4, pp. 55–68 (2008).
[18] Gueye, M., Sarr, I. and Ndiaye, S.: Database Replication in Large Scale Systems: Optimizing the Number of Replicas, *Proceedings of the 2009 EDBT/ICDT Workshops*, pp. 3–9 (2009).
[19] Bonvin, N., Papaioannou, T. G. and Aberer, K.: A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage, *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 205–216 (2010).
[20] Shen, H.: An Efficient and Adaptive Decentralized File Replication Algorithm in P2P File Sharing Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 21, No. 6, pp. 827–840 (2010).
[21] Chang, R.-S., Chang, H.-P. and Wang, Y.-T.: A dynamic weighted data replication strategy in data grids, *IEEE/ACS International Conference on Computer Systems and Applications(AICCSA)*, pp. 414–421 (2008).