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

2005－DPS－122（32）
2005－CSEC－28（32）
2005／3／23

分散環境における順位つき資源への部分探索法の提案

添田祐司†，泉泰介†，増澤利光†

分散システム内に分散配置された資源の検索にかかるコストを小さくする方法として部分検索法が提案されている．しかし，既知の部分検索法では，利用者の要求に対して検索結果の品質が考慮されていないという問題を持つ．そこで本稿では，分散システム内の資源に順位を付加したモデルを考え，資源の順位を考慮した部分検索法を定式化する．また，分散システム上でそれを実現する方法として，確率的配置法を提案する．確率的配置方法は，各サーバが資源を，その順位に基づく保持確率で保持する手法であり，システムの状況変化に対する高い適応性を有する．本稿では，低検索コストを実現する確率的配置法について考察を行い，シミュレーションによって評価を行う．また，保持確率関数が与えられたとき，検索にかかるコストを悪化させることなく，順位の変動等によるシステム更新にかかるコストを改善するような保持確率関数へと変換する方法を提案し，その有効性をシミュレーションによって示す．

Partial Lookup Services for Distributed Resources with Ranks

Yuji Soeda†，Taisuke Izumi†，Toshimitsu Masuzawa†

Partial lookup is a method to retrieve distributed resources. For a query, it can return any subset of all resouces matching the query. While the partial lookup can achieves lower cost than the total lookup, it does not guarantee quality of lookup results. In this paper, we introduce rank of resouces, and formalize the partial lookup for ranked distributed resources. In addition, we proposes a probabilistic method to realize the partial lookup for ranked resouces in distributed systems. In this method, each server stores resouces with probability according to their ranks. The probabilistic method has adaptability to system dynamics since it requires no inter-server communication. We show the efficiency of this method by simluations. We also propose probability function transformation to improve efficiency for system dynamics and show that the transformation introduces no degradation of lookup cost.

1　はじめに

複数の計算機をネットワークで相互接続した分散システムにおいて，システム内に存在する資源の中から，利用者が要求する資源を検索，取得することは分散検索とよばれ，基本的な機能の一つとして重要である [1, 2].

一般に，従来の分散検索システムは，クエリに適合する資源を可能な限り網羅的にユーザへと返すことを目的としている．そのため，検索対象となる資源の大規模化に対して，検索時間，および通信コスト等において，スケーラビリティを確保することが困難である．近年，この問題に対処する方法として，Sunらにより，部分検索と呼ばれる方法が提案されている [3]．部分検索は，要求する結果の個数をクエリに付加することにより，クエリに適合する資源のうち，要求された個数だけ，検索の結果として返す．Sunらは，分散検索システム上で部分検索のための資源配置の方法をいくつか提案し，シミュレーションによる評価を行っている．この結果では，クエリ問い合わせにかかるコスト (e.g. 時間，通信量)，あるいはシステムにかかる負荷，故障耐性等において従来手法より優れていることが示されている．一般に，現実の検索では，多くの場合において，ユーザが本当に必要とする資源はクエリに適合する資源のごく一部であることが多く，部分検索手法は，ユーザの利便性を損なうことなく分散検索システムを高性能化する手法として有用である．

しかし，Sunらの部分検索は，クエリに適合する資源であれば，いかなる資源を返しても良いと定義されている．しかし，実際の検索では，ユーザはクエリに適合する資源のうちでも，より適合度の高いデータを結果として知りたいという要求を持つことが多い．そのため，単純に適合資源の一部を返すだけでは，ユーザが望む資源をシステムが提供できないことがしばしば起ると考えられる．

そこで本稿では，資源に順位を導入し，より順位の高いデータから優先的に部分検索の結果として返すことを目的とした，分散検索システムの実現について考える．本稿ではまず，従来の部分検索法を，順位を導入した資源を取り扱うように拡張した部分検索法を定義する．さらに，分散検索システム上での順位つき資源の部分検索のための資源配置法を提案し，シミュレーションで評価する．

2　諸定義

2.1　分散検索システム

分散検索システムとはシステム内に存在する資源の検索機能をユーザに対して提供するシステムである．ここで資源とは，テキストファイルやプログラ

†大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

ムなどのデータ一般を抽象化したものである．本稿で考える分散検索システムはクライアントと，システムを構築するサーバ集合 S とシステム検索対象である順位つき資源の集合 V から成る．

クライアントは，ユーザからの検索要求を受け付け，システム内に存在する資源に対して検索を行う．その後，要求を満たす資源を結果として返す．

サーバ集合 S は n 台のサーバで構成される．各サーバはそれぞれ資源を保持し，クライアントの問い合わせに応じて資源を返す．また，各サーバは V 内の任意の資源を保持できるものとする．ただし，各サーバが保持できる資源の数には上限 $limit$ がある．各サーバは固有の識別子 $i \in \{0,1,\ldots,n-1\}$ をもち，サーバ i が保持している資源の集合を R_i で表す．

分散検索システム内の資源 $v(\in V)$ には順位が割り当てられている．各資源は上位から $1,2,3,\ldots,|V|$ という順位を持つ．資源 v に対する順位を $f(v)$ で表す．形式的には，f は任意の異なる資源 u,v について $f(u) \neq f(v)$ を満たす，資源集合 V から集合 $\{i \in N | 1 \leq i \leq |V|\}$ への写像 $f : V \rightarrow \{i \in N | 1 \leq i \leq |V|\}$ である．ただし，ここで N は自然数の集合を表す．資源集合 V および写像 f は資源の追加や削除，順位変動などの発生によって変化しうる．

分散検索システムの状況は，システム内の各サーバが保持する資源集合の組 $(R_0, R_1, \ldots, R_{n-1})$ で表される．システムの状況はイベントが生起することにより遷移する．一般性を失わずシステム全体で同時に一つのイベントしか生起しないと仮定できる．状況とイベントの交互列 $c^0, e^1, c^1, e^2, \cdots$ (c^i, e^iはそれぞれ状況，イベントを表す) を分散検索システムの実行と定義する．ただし，e^iは c^{i-1}で生起可能なイベントであり，c^{i-1}で e^iが生起後の状況が c^iである．ここで，c^0は分散検索システムの初期状況である．状況 c^tにおける資源集合 V，写像 f，各サーバ i の保持する資源集合 R_iをそれぞれ V^t，f^t，${R_i}^t$と表す．

システムで生起するイベントとして次の 2 種類のイベントを定義する．V および f はこれらのイベントによって変化しうる．それぞれのイベントが $e^t (t \geq 1)$ として生起した場合の動作を用いて説明を行う．

- $add(d, rank)$
  資源 d を $rank$ 位の資源として V に追加するイベント．直前の状況 c^{t-1} において $rank$ 位以下であった資源の順位は 1 位ずつ繰り下がる．
- $delete(d)$
  資源 d を資源集合 V から削除するイベント．直前の状況 c^{t-1}において順位 $f^{t-1}(d)$ より下位の資源は順位が 1 位ずつ繰り上がる．

2.2 順位つき資源における部分検索

本稿では，部分検索システム [3] を拡張した，順位つき資源を対象とした部分検索システムを考える．順位つき資源に対する部分検索システムは，$partial\_lookup(t, \alpha)$ と呼ばれる操作をユーザに対して提供する．ここで αは 1 以上の実数である．$partial\_lookup(t, \alpha)$ による問い合わせが生起したとき，システムは V において順位がαt より上位であるαt 個の資源から，t 個の資源を問い合わせ元へと返す．返される t 個の資源は，その順位がαt よりも上位である限りどれでもかまわない．以降，ある問い合わせ $partial\_lookup(t, \alpha)$ に対して，t を要求数，αを緩和率と呼ぶ．

n 台のサーバ集合からなる分散検索システム上に順位つき部分検索システムを実現することを考える．部分検索の検索コストは，検索完了までにアクセスしたサーバ台数と定義する．また，サーバへと問い合わせることで，クライアントが，条件を満たす資源を要求数以上取得できたとき，その検索は成功したという．一方，全サーバに問い合わせても，クライアントが，条件を満たす資源を要求数だけ取得できなかったとき，その資源要求は失敗したと呼ぶ．失敗した問い合わせの検索コストはシステム内の全サーバ数 n と定義する．

3 確率的配置方法

3.1 資源配置方法

順位つき資源の部分検索において，検索コストのみを単に最適化することを考えた場合，上位の資源から順に各サーバに割り当て，検索を行うときは上位の資源を有するサーバから順に問い合わせを行えば，最適な検索コストを達成することが可能である．しかし，この手法では，順位が上位の資源を有する特定のサーバへの負荷集中が生じる．さらに，データの順位が変化した場合における更新コスト，あるいはシステムを構築するサーバが故障した場合における資源の再配置に多大なコストを要するため，この手法は現実的ではない．

そこで，本稿では低検索コストと低更新コストの両立を目的とした確率的配置方法について述べる．部分検索における確率的配置方法は，以下の特徴を有する．

- サーバをランダムに選択してアクセスを行うことでの負荷分散が実現できる
- 各サーバでの資源配置方法の独立性が高く，サーバの故障や参加が起こる環境に適している

提案する確率的配置方法では，各サーバは各資源をその順位に応じて決まる確率で保持する．一般に，順位が上位の資源は高い確率で，逆に順位が下位の資源は低い確率でサーバに保持される．クライアントは，必要な資源数を取得できるまで，サーバ集合からランダムに一つずつサーバを選び，問い合わせ

を行う．確率的配置手法を利用した場合，上位の資源は高い確率で多くのサーバに保持される．その一方で，各サーバの保持する資源が上位の資源のみからなる確率は低く，下位の資源もある程度の割合で保持されることが期待できる．そのため，要求数が小さい問い合わせに対しては，問い合わせ先によらず少ない検索コストで完了でき，要求数が大きい問い合わせに関しては，複数のサーバへと問い合わせることで，(検索コストは上がるが) 検索を成功させることができると期待できる．また，確率的配置方法では各サーバが独立に処理を行うため，サーバの増減，資源集合の状況変化時にサーバ間の協調処理を必要としない．さらに，クライアントは各サーバを区別する必要はないので，全サーバ集合を正確に知らなくとも問い合わせ処理が可能である．そのため故障等が生じサーバ数の変化がおきるような動的環境に適する手法であるといえる．

以下，確率的配置方法の詳細について述べる．なお，以降の説明で，各サーバが順位 i の資源を保持する確率を $Q(i)$ で表し，保持確率関数と呼ぶ．保持確率関数 $Q(i)$ は以下を満たす．

$$\forall i(1 \leq i \leq h) : 0 \leq Q(i) \leq 1 \quad (1)$$

$$\forall i(h < i \leq |V|) : Q(i) = 0 \quad (2)$$

$$\sum_{i=1}^{h} Q(i) = EXP \quad (3)$$

式 (2) は，システムに保持する資源の種類が高々 h 種類であることを示している．以後，h を資源の上限値と呼ぶ．式 (3) は，各サーバが保持する資源数の期待値が EXP であることを示している．一般的に値 EXP は，サーバの記憶領域を有効に利用するために，$EXP = limit$ と設定すればよい．ただし，$Q(i)$ に従って各資源を保持するかどうかを決定すると，実際に保持すべき資源の数が上限 $limit$ を超える場合がある．そのときは，いくつかの資源を確率的に削除することで，保持する資源の個数を $limit$ 個に削減する．具体的には，以下のような処理を行うことで保持するべき資源の数を削除する．以後，この処理を $Overflow$ と呼ぶ．

1. V_s : サーバが保持することになった資源の集合とする．
2. $|V_s| \leq limit$ ならば処理を終了する．
3. 確率 $\frac{1-Q(v)}{\sum_{k \in V_s}(1-Q(k))}$ で資源 $v \in V_s$ を 1 つ選択し，V_s から削除
4. 2 に戻る．

3.2 資源の変化による更新

分散検索システムでは，資源の追加，削除により，資源集合や資源の順位が変化しうる．このとき，各サーバにおける順位 i の資源の保持確率を $Q(i)$ に保ち続けるためには，確率的配置方法では，資源の変化に応じて資源の再配置を行う必要がある．本節では，資源の内容変化が起きたときの更新処理について述べる．

確率的配置における更新処理の目的は，更新前後において，サーバが各資源を保持する確率を保持確率関数 $Q(i)$ に従わせることである．そのため，実際の更新処理では，更新による順位変動が起るすべての資源について，確率的な再配置を行う必要がある．

具体的な処理は以下のとおりである．以下では更新前後の状況 c^{t-1} と c^t の差異を用いて説明を行う．

- $Q(f^{t-1}(v)) \neq Q(f^t(v))$ であるようなすべての $v \in V$ について以下の処理を行う．
  - $Q(f^{t-1}(v)) > Q(f^t(v))$ の時
    - $v \in R_i$ であるサーバ i : $1 - \frac{Q(f^t(v))}{Q(f^{t-1}(v))}$ の確率で v を削除する．
    - $v \notin R_i$ であるサーバ i : 何もしない．
  - $Q(f^{t-1}(v)) < Q(f^t(v))$ の時
    - $v \in R_i$ であるサーバ i : 何もしない．
    - $v \notin R_i$ であるサーバ i : $\frac{Q(f^t(v))-Q(f^{t-1}(v))}{1-Q(f^{t-1}(v))}$ の確率で v をサーバ i が保持する資源集合 R_i に追加する．

上記の更新処理に対して $limit$ が無限大の場合 (つまり，$Overflow$ 処理が生じない場合)，以下の定理が成り立つ．

定理 1 初期状況 c^0 において，サーバ i が確率 $Q(x)$ で順位 $x(= f^0(v))$ の資源 v を保持しているならば，任意の状況 $c^t(1 \leq t)$ においてサーバ i は確率 $Q(x)$ で順位 x の資源を保持する．

4 シミュレーション評価

本節では，前節で提案した確率的配置手法について，シミュレーションにより評価を行う．

4.1 関数の選択

確率的配置手法において，保持確率関数 $Q(x)$ の選択は，システムの性能に大きな影響を与える．本節では，いくつかの保持確率関数に対して，シミュレーションによる性能評価を行う．

4.2 保持確率関数

シミュレーションで用意する保持確率関数は，ある確率関数 $P(x)$ から求める．確率関数 $P(x)$ は以下の条件を満たす関数 $P(x)$ である．

$$\forall i(1 \leq i \leq h) : 0 \leq P(i) \leq 1 \quad (4)$$

$$\forall i(h < i) : P(i) = 0 \quad (5)$$

$$\sum_{i=1}^{h} P(i) = 1 \quad (6)$$

直感的には，確率関数 $P(x)$ は各順位の資源を保有する確率を，その総和が 1 になるように標準化したも

のと言える．順位 i の資源の存在確率がほぼ $P(i)$ に比例し，各サーバが保持する資源数の期待値が EXP となる保持確率関数 $Q(x)$ を次の手続き $P2Q$ によって求める．

[手続き $P2Q$]
以下を満たすように $\gamma(\geq EXP)$ を設定する．

- $Q(x) = min(\gamma P(x), 1)$
- $\sum_{i=1}^{h} Q(x) = EXP$

4.3 評価対象とする関数

3 種類の保持確率関数 $Random$，$RankBase1$，$RankBase2$ に対してシミュレーションを行った．これらは以下の確率関数から手続き $P2Q$ により導き出したものである．

[確率関数 $Random$]
確率関数 $Random$ は，各サーバが資源 $v \in V(1 \leq f(v) \leq h)$ を順位に関係なく等確率で保持する関数である．確率関数 $P(x)$ は以下の式で定義される．

$$P(x) = \begin{cases} \frac{1}{h} & (1 \leq x \leq h) \\ 0 & (h < x \leq |V|) \end{cases}$$

[確率関数 $RankBase1$]
確率関数 $RankBase1$ は，資源の保持確率が順位の低下に応じて線形的に減少する関数である．確率関数 $P(x)$ は以下の式で定義される．

$$P(x) = \begin{cases} \frac{h-x+1}{\sum_{k=1}^{h} k} & (1 \leq x \leq h) \\ 0 & (h < x \leq |V|) \end{cases}$$

[確率関数 $RankBase2$]
確率関数 $RankBase2$ は，$RankBase1$ と同様，順位の低下に応じて確率が減少してゆく関数である．ただし，$RankBase1$ とは異なり，下位の資源になるほどその近傍における確率の差は小さくなる．確率関数は以下の式で定義される．

$$P(x) = \begin{cases} \frac{1/\sqrt{x}}{\sum_{v \in V} \frac{1}{\sqrt{f(v)}}} & (1 \leq x \leq h) \\ 0 & (h < x \leq |V|) \end{cases}$$

4.4 シミュレーション結果

4.5 関数の選択による影響

保持確率関数として $Random$，$RankBase1$，$RankBase2$ を利用した場合の検索コストについて，シミュレーションにより比較を行う．検索コストとは検索完了までにアクセスしたサーバの数である．シミュレーションの設定は，$n = 10$，$h = 1000$，$limit = \infty$，$EXP = 200$，$\alpha = 1.5$ としている．シミュレーション設定において，$limit = \infty$ としているのは，処理 $Overflow$ による影響がない状態で，保持確率関数そのものの特性について議論するためである．

図 1 は，要求数の変化に対する各保持確率関数の検索コストの変化をプロットしたもので，横軸は検索における要求数，縦軸はそれに対する検索コストである．プロットされているデータは各保持確率関数について 50 回の試行を行った平均値である．失敗した検索は $n = 10$ としてプロットしている．

シミュレーション結果より，t と h の割合 t/h に対して，次のことがわかる．t/h の値が 0 から約 0.1 にかけては，$RankBase1$ の検索コストが最も良い．また，t/h の値が約 0.2 から約 0.6 にかけては $RankBase2$ の検索コストが最も良く，t/h の値が約 0.6 から 1 にかけては $Random$ の検索コストが最も良い．なお，同様のシミュレーションをパラメータ $limit$，α の値を変化させても行っているが，上記の t/h による確率関数の優位性は，$limit$，α の値により t/h の区間に多少の変動があるものの，同じ傾向を示す．

$RankBase1$，$RankBase2$ において，検索が成功するのは t/h が 0.8 程度までとなっている．しかし，通常起こりうる要求数 t に対して h は十分大きい値をとるため，$RankBase1$，$RankBase2$ は現実的な問い合わせの範囲では検索コストが小さく有効である．

4.6 パラメータ変化による影響

本節では，システムのパラメータ変化によるシステムへの影響についてシミュレーションで評価を行う．なお，本節では $RankBase1$ のみをシミュレーションの対象としているが，$Random$，$RankBase2$ に対しても同様の結果が得られる．

[緩和率 α についての性質]
緩和率 α を変更した場合の性質を示す．シミュレーションの設定は，$n = 10$，$h = 1000$，$limit = \infty$，$EXP = 300$ としている．ただし，α は 1.0，1.5，2.0 の 3 種類である．

α の変化に対する結果を図 2 に示す．図 2 は保持確率関数 $RankBase1$ に対して，α を変化させた場合の検索コストを比較した結果であり，図 1 と同様の方法でプロットしている．

結果より，緩和率 α が大きくなるほど検索コストが低下していることがわかる．これは，緩和率 α を大きくすることで，1 台のサーバにアクセスしたときに，条件を満たす資源をより多く得られるためである．この傾向はすべての保持確率関数について成立している．よって，ユーザが部分検索を行うときに緩和率 α をうまく設定することによって低コストな検索で要求を達成できることがわかる．

[$limit$ の変化に対する影響]
サーバが保持できる資源数 $limit$ を変化させたときの性質を示す．$limit$ を有限にした場合，$limit$ が無限のときには行われなかった処理 $Overflow$ が行われる．そのため，サーバが各資源を保持している確率は，定められた保持確率関数と若干異なるものになり，検索コストにも影響が出ると考えられる．

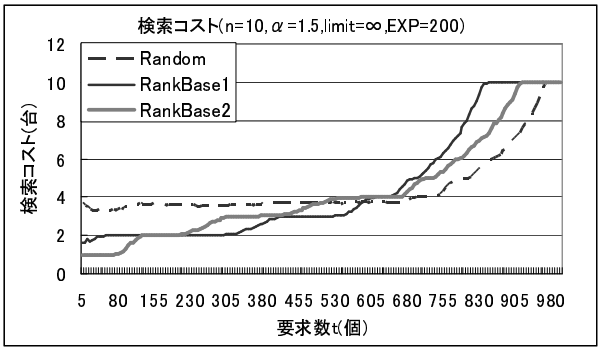

図 1: 検索コスト：保持確率関数の影響

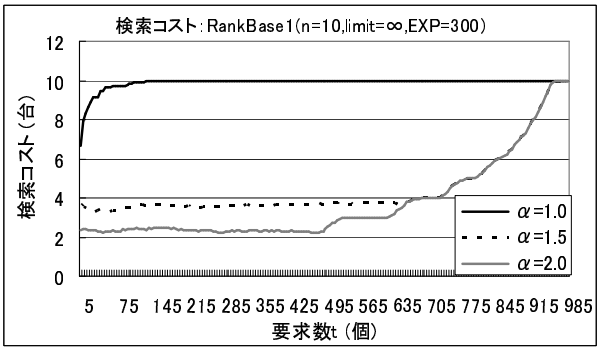

図 2: 検索コスト：αの影響

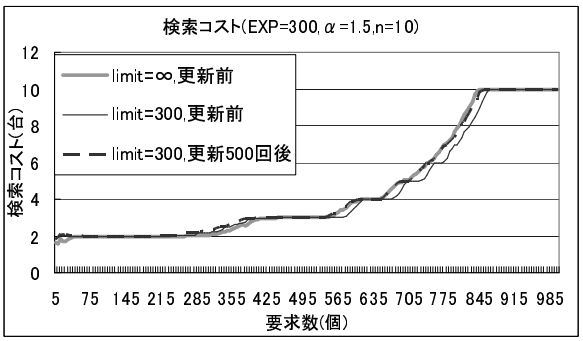

図 3: 検索コスト：$limit$の影響

また，3.2節で，保持確率を維持する更新処理を示しているが，そこでは$limit$が無限であることを仮定していた．$limit$が有限の場合は，処理$Overflow$が行われるため保持確率を維持することが保証されているわけではない．そこで，$limit$が有限の場合において，更新処理が検索コストに与える影響をシミュレーションによって示す．

シミュレーションの設定は，$n = 10, h = 1000$, $limit = 300 or \infty, EXP = 300, \alpha = 1.5$としている．ただし，$limit$は有限と無限の2種類であり，有限の場合の値は$limit = 300$である．また，更新処理は$add$,$delete$をランダムに選択しランダムな順位の資源に対して合計500回行う．

$limit$の変化に対する結果を図3に示す．図3は保持確率関数$RankBase1$に対して，$limit$を変化させた場合の検索コスト，更新処理を行う前と更新処理を500回行った後の検索コストを比較した結果であり，図1と同様の方法でプロットしている．

結果より，$limit$が無限の場合と有限の場合で検索コストはほぼ等しく，同じ傾向を持つ．この傾向はすべての保持確率関数について成立している．従って，$limit$を無限から有限に変更しても問題は生じない．

また，更新処理の前と後での検索コストは$limit = \infty$の場合とほぼ等しい．つまり，$limit$が有限の場合における更新処理も目的とする保持確率関数をほぼ維持した更新が行えることがわかる．この傾向はすべての保持確率関数について成立している．従って，$limit$を有限にした場合においても更新処理は有効である．

5 更新コストを低下する方法

前節では，現実的な範囲の問い合わせについては$RankBase1$や$RankBase2$が有効であると述べた．一方，更新時には，保持確率が変わる資源に対して再配置処理が必要となり，$RankBase1$や$RankBase2$は$Random$に比べて更新コストが大きいという問題を持つ．そこで本節では，更新コストの大きい保持確率関数$Q(x)$に対して，更新コストを低下させるために$Q(x)$を変換するアルゴリズム$Step$を提案する．また，アルゴリズム$Step$の有効性をシミュレーションによって示す．

アルゴリズム$Step$での保持確率関数の生成方針は次のとおりである．更新処理は順位変動が発生した場合に，順位変動前と順位変動後の保持確率が異なった場合に行われる(3.2節参照)．そのため，更新コストの低下を図るためには，順位変動前と順位変動後の保持確率が異なる資源数を減らせばよい．そこでグラフ化した場合に階段状のグラフを形成するステップ型関数$S(x)$に変換することが考えられる．$Q(x)$から$S(x)$への変換はアルゴリズム$Step$によって行われる．$Step$は入力として$Q(x)$以外に整数stpを取り，$stp+1$段以下の段数からなる関数$S(x)$を出力として返す．つまり，1回の順位変動が起こったときに保持確率が変化する資源数は最大$stp+1$個であり，1台のサーバにおける順位変動1回当たりの更新コストの上限が$stp+1$となる．よって，アルゴリズム$Step$を用いて$Q(x)$を$S(x)$に変換することで更新コストの低下を実現することが期待される．具体的なアルゴリズム$Step$の説明は以下のとおりである．

- $sep = limit/stp$とする．
- h位の資源から順に次の条件を満たすように順位

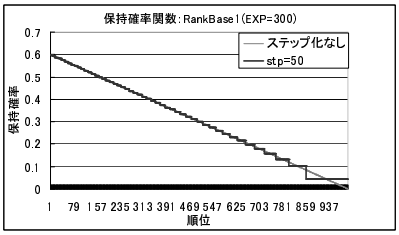

図 4: 保持確率関数：$RankBase1$

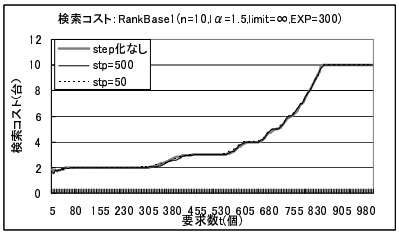

図 5: 検索コスト：ステップ化の影響

図 6: 更新コスト：ステップ化の影響

stp	$RankBase1$	$RankBase2$
なし	244.74	245.02
500	65.63	74.84
50	8.52	9.18

$x(0 \leq x \leq h)$ を順位の集合 $R_i = \{p, ..., q | p \leq k \leq q(k \in R_i)\}(0 \leq i \leq stp)$ に分割．

$$(\sum_{k \in R_i} Q(k) - Q(p)) < sep < \sum_{k \in R_i} Q(k) \quad (7)$$

- 各 $R_i(0 \leq i \leq stp)$ に属する順位 $x \in R_i$について $S(x)$ を次のように設定する．

$$S(x) = \frac{\sum_{k \in R_i} Q(k)}{|R_i|}$$

以上の操作を $Q(x)$ に対して行うことで，ステップ型関数 $S(x)$ を得る．得られた $S(x)$ は $stp + 1$ 段の段から構成される．また，入力である保持確率関数 $Q(x)$ が単調減少関数であることと，$S(x)$ における段は不等式 (7) の条件で作成されるため，$S(x)$ も単調減少関数となり，$S(x)$ も検索コストに関して低コストを実現することが期待される．

以上の $Step$ を用いて $RankBase1$ や $RankBase2$ をステップ型に変換した関数についてシミュレーションを行い更新コストを比較する．また，検索コストに与える影響に関しても調べる．シミュレーション設定は，$n = 10, h = 1000, limit = \infty, EXP = 300$, $\alpha = 1.5$ としている．

保持確率関数 $RankBase1$ に対してステップ化を行った結果の保持確率をそれぞれ図 4 に示す．また，$RankBase1$ と $RankBase2$ をステップ数 st を変えて変換した場合の更新コストを表 6 に示す．表 6 からわかるように，ステップ数 st を小さくすればするほど更新コストを低下させることができる．次にステップが，検索コストに対して与える影響をシミュレーション結果によって評価する．結果は図 5 である．グラフにおいてステップ化を行った場合の影響はあまり見られない．

以上より，ステップ化アルゴリズム $Step$ は検索コストに関する特性をほとんど変えずに更新コストを大きく低下させることができる．つまり，確率的配置方法において，ステップ型の関数が更新コストに優れている分，$RankBase1$ や $RankBase2$ などに代表される連続関数より優れているといえる．

6 まとめ

本稿では，分散検索システムにおける順位つき資源に対する部分検索を定義し，それを実現する確率的手法を提案した．確率的配置方法では，順位の変動が発生する動的なシステム上で適切な保持確率を維持するために必要な更新処理を提案した．また，いくつかの保持確率関数について，シミュレーションにより検索コストの評価を行った．さらに，動的なシステムに対する確率的配置の適応性を高めるために，更新コストを低下させるように保持確率関数を変換するアルゴリズムを提案し，シミュレーションによりその有効性を示した．

今後の課題として，以下のことがあげられる．

1. 最適な検索コストを実現する保持確率関数を求めること．
2. クエリによって資源の順位が異なる，より一般的な順位つき資源に対する部分検索法について考察する．
3. サーバの容量やアクセスコストに多様性を有する，より一般的な分散システム上への適用を考察する．

参考文献

[1] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48, 2003.

[2] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, 2001.

[3] Qixiang Sun and Hector Garcia-Molina. Partial lookup services. In *Proceedings of the 23rd International Conference on Distributed Computing Systems(ICDCS)*, page 58. IEEE Computer Society, 2003.