複数のRaspberry Piを用いた分散システムの構築

複数の Raspberry Pi を用いた分散システムの構築

小澤卓也
(静岡大学技術部情報部門)

1. はじめに

静岡大学情報学部では、情報科学科 3 年生向けの実験科目「情報科学実験 A」が開講されている。この実験科目ではソケット通信をはじめ、HTTP サーバの構築・データベースを含むシステムの実装など、ネットワークシステムに関わる幅広い技術を学ぶことができる。その実験科目の最終課題として、Web 3 層構造によるデータ検索システムを構築し、高速化を試みる課題がある。

一方で複数の計算機によって処理を高速化させるという考え方は以前から存在する。アンドリューらによると、1980 年代中頃の、高性能マイクロプロセッサの開発と高速コンピュータネットワークの発明により、現在では多数のコンピュータを高速ネットワークによって接続し、簡単に分散システムとして構築できるようになった、としている[1]。また George らは分散システムの持つ重要な特性として、並行性・規模透過性・可用性を持つと述べている[2]。

そこで筆者は実験科目の最終課題に対し、分散システムの並列性に着目して高速化を試みた。比較的安価で入手できる Raspberry Pi を 7 台用いて、高速に検索処理を行うシステムを構築した。本稿ではその手法について報告する。

2. 実装した画像検索システム

2.1 システム概要

本実験では、Web ブラウザで検索キーワード (Tag) を入力すると、その Tag がつけられた画像を撮影時刻が新しい順に最大 100 件表示するアプリケーションを実装した (図 1)。このシステムは、クライアント・アプリケーションサーバ (AP サーバ)・データベースサーバからなる 3 層構造となっており、各層にはそれぞれ異なる役割を持つ。クライアントでは、Web ブラウザ上で、検索用の入力フォームと検索結果を表示する。ユーザは検索したい画像の Tag 文字列をそこで入力し、AP サーバに検索を要求する。AP サーバでは、クライアントから受け取った Tag を元にデータサーバに適切な検索要求を発行し、検索結果を取りまとめる。最後にクライアントに返すページを生成してクライアントに返す。

データサーバでは検索の元となるデータを管理し、AP サーバからの検索要求に応じて実際に検索を実

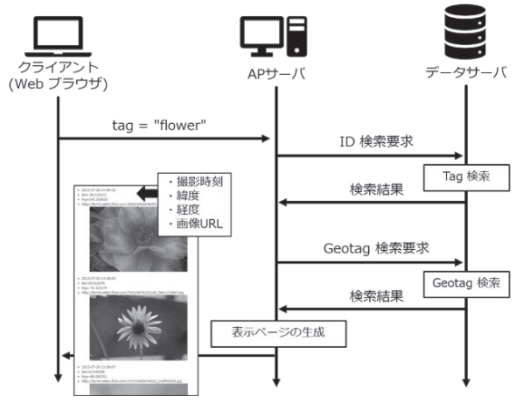

図 1. クライアント・AP サーバ・データサーバの 3 層構成による検索システム

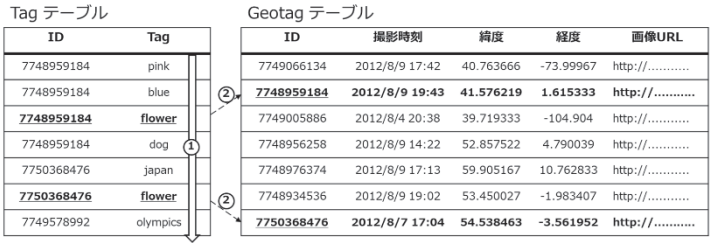

図 2. データサーバ上のデータから画像を検索する手順

図 3. サーバ機となる Raspberry Pi 3 Model B

表 1. Raspberry Pi 3 Model B の仕様 [3]

項目	スペック
CPU	Quad Core 1.2GHz Broadcom BCM2837 64bit
RAM	1 GB
Ethernet	100 Base Ethernet
USB	4 USB-2 port
GPIO	40-pin extended GPIO
Storage	Micro SD

行する。画像に対する Tag と撮影情報は、それぞれ Tag テーブルと Geotag テーブルに列挙されている。Tag テーブルには、写真につけられた固有の番号 (ID) とその写真につけられた Tag が対応付けられ、列挙されている。Geotag テーブルには、ID に対して撮影情報 (撮影時刻・緯度・経度・画像 URL) が対応付けられて列挙されている。

検索時にはまず、ユーザが検索したい画像の Tag をクライアント上で入力し、AP サーバに送信する。Tag を受け取った AP サーバはデータサーバに対し、ユーザが入力した Tag に合致する ID を Tag テーブルの中から全て列挙させる要求を送る。データサーバでの ID 検索結果をもとに、列挙した ID それぞれについて Geotag テーブルの ID を参照しながら撮影情報を検索する。これにより、ユーザが指定した Tag をもつ画像の撮影情報のリストが得られる。最後にリストを撮影時刻の新しい順に並び替え、先頭から最大 100 件の撮影情報を返す。

2.2 使用機材

今回はサーバ機として Raspberry Pi 3 Model B Rev 1.2 を使用した (図 3)。この計算機に Raspbian GNU/Linux 11 をインストールし、AP サーバ・データサーバをそれぞれ実装した。本番環境ではこの計算機を複数台用意し、検索用サーバとして稼働するようにした。Raspberry Pi の仕様を表 1 に示す。また、Raspberry Pi の OS は Raspbian GNU/Linux 11 を利用した。

3. サーバの構築

3.1 開発環境

複数の計算機に展開するシステムを効率的に開発するにあたり、課題が 2 点あった。1 つは本番環境で試す前に簡単なデバッグを行えるテスト環境が必要だった点。もう一つは、プログラムが完成して本番環境に適用する際、複数の計算機に最新のコードを移す手間や時間がかかる点である。

この問題を解決するため、バージョン管理システム (GitHub) を利用したクロス開発を行った (図 4)。開発時には仮想環境 (Ubuntu 20.04) 上でコード作成・デバッグを実施した。また、コードを適宜 GitHub に Push しながら開発を進めた。開発状況が一区切りつき、本番環境で試す際には、構成する各計算機で最新のコードを GitHub から取得してビルドすることとした。

この方法によって、開発上いくつかの利点が生まれた。実機と比べて気軽にテストできる環境を整えたことで、プログラムのテストを細かく実施できた。またバージョン管理システムによって、失敗しても元に戻せる安全な環境下で作業できた。更に、最新のコードを本番環境へ取り込む際は、各計算機上で 2 コマンド ($ git pull; $ make;) を実行すればよいレベルまで簡略化できた。

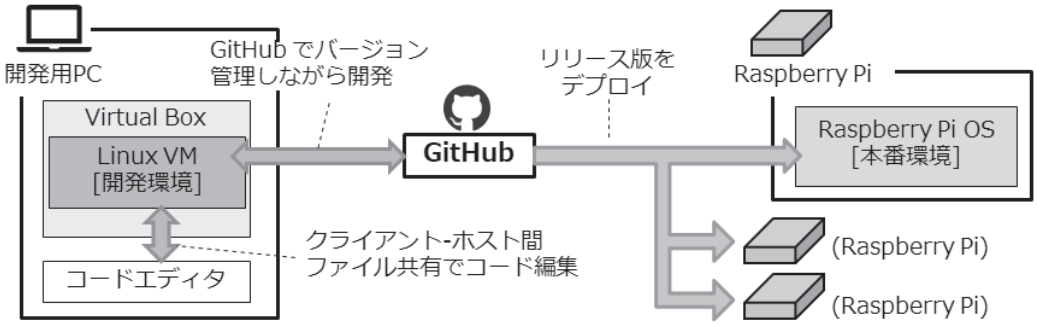

図 4. 開発環境と本番環境の構成

3.2 システム構成の検討

この検索システムは大きく分けて、1) Tag をもつ ID 全てを検索するタスクと、2) ID から撮影情報を検索するタスクの 2 つのタスクに分けられる。そこで、これらのタスクを処理するサーバをそれぞれ複数の計算機で構築し、ネットワークを介する通信によってサーバ群同士で互いに必要な情報を送受信することとした (図 5)。前者のタスクは ID 検索サーバ、後者のタスクは撮影情報検索サーバが担うこととし、それぞれのサーバの構成は各タスクの計算量やメモリ使用量を元に決定した。

ID 検索サーバではメモリ効率と探索の計算量を考慮し、データを木構造でメモリ上に展開することとした (図 6)。Tag をキー、ID のリストを値とする構造のため、テーブルによる表現と比べて Tag 文字列が重複しない分 (図 6 の灰色部分) メモリ効率が良い。検索時には木構造の root から Tag をたどり、Tag が見つかった際は ID のリストを返す。このときの計算量は Tag データの総数 N_T に依存し、検索時の計算量は $O(\log_2 N_T)$ となる。一方で、元の CSV ファイルが 400 MB 近くのデータ量であることと、Raspberry Pi のメモリサイズが 1 GB であることを考慮し、余裕をもって機器 2 台にデータを分散させて管理するようにした。検索時には 2 台の ID 検索サーバに要求を送り、AP サーバで各サーバの結果を受け取りと結合を行うこととした。

撮影情報検索サーバでは、元の CSV ファイルのサイズが 1.1 GB あるため、すべてのデータをメモリに

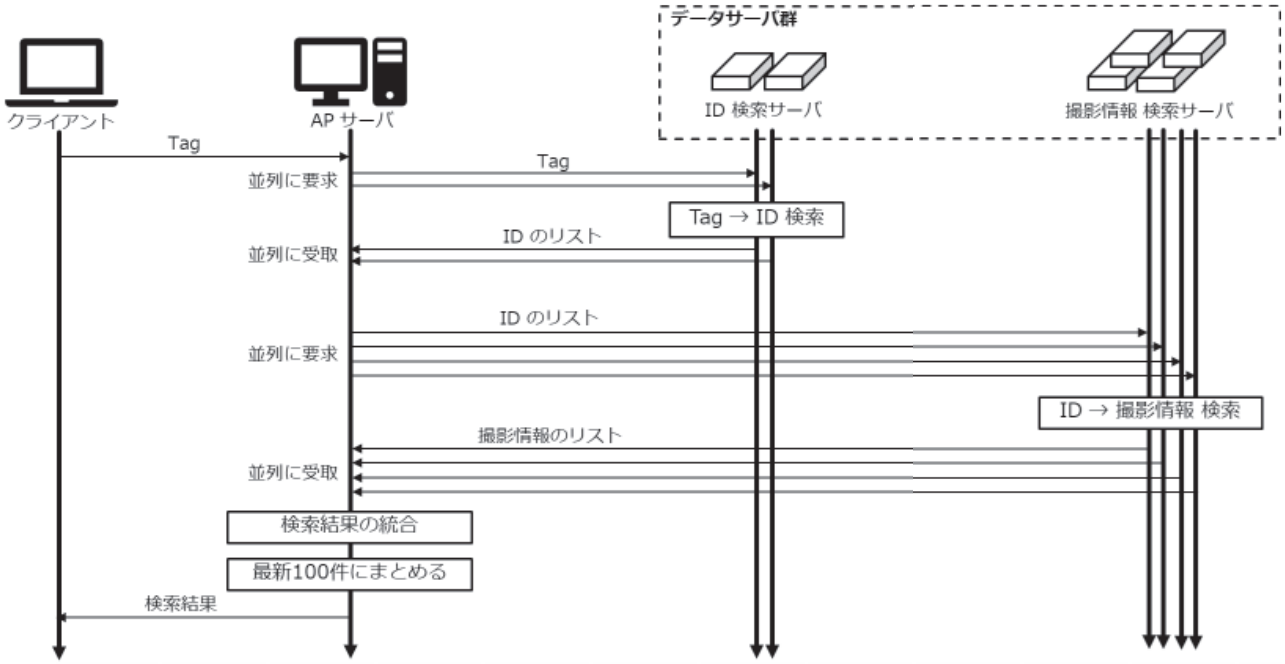

図 5. データサーバを複数計算機で構成したシステムにおけるシーケンス図

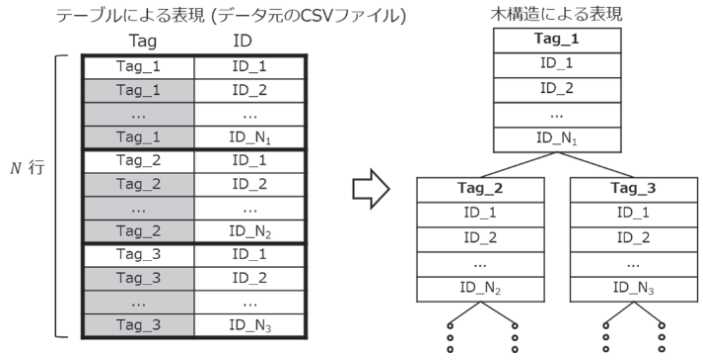

図 6. 木構造を利用した Tag のデータ構造

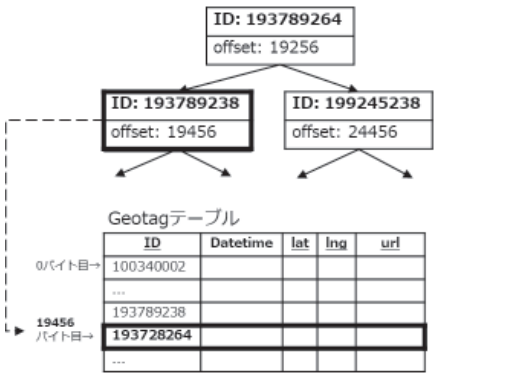

図 7. 木構造を利用した Geotag のデータ構造

載せることはできなかった。そこで、ID に対応するデータが元データの何バイト目にあるかというデータ (オフセット) を木構造にしてメモリ上に展開することとした (図 7)。検索時にはまず木構造をたどってオフセットを取得する。このオフセット分ファイルの先頭からシークして 1 行分読み込むことで、撮影情報を取り出すこととした。撮影情報を取得するための計算量は、要求される ID の個数 M と Geotag データの総数 N_G に依存し、$O(M \log_2 N_G)$ となる。ID 検索サーバと比べて計算量が多いため、こちらは 4 台構成とした。

4. 性能の調査

4.1 実験方法

実際にシステムを実装し、性能の調査を行った。比較対象として、提案手法の ID 検索サーバ・撮影情報検索サーバのプログラムを AP サーバ上で 1 プロセスずつ動かすシステムを用意し、1 台の計算機で実行する単一構成での性能も調査した。また、同様の検索アルゴリズムをデータベース (DB) と SQL によって実装し、インデックスによる高速化を適用したシステムと適用しないシステムを用意した。

提案手法を含むこれらの 4 つの実装方式で、クライアントから Tag を送信してから応答が返ってくるまでの時間を計測した。この応答時間を比較することで、性能の比較を行った。

4.2 実験結果

計測結果と各システムの実行時間を比較したグラフを示す (表 2, 図 8)。この結果を見ると、DB による実装と比べて提案手法は単一構成・複数構成いずれも高速に動作した。また、提案手法の中でも、単一構成と比べて複数構成の方が 10 倍以上高速だった。一方で Tag に「raspberrypi」を指定した場合など、ヒットした件数が少ない場合は単一構成で検索する方が高速だった。また、Index なしの DB を利用する方法で「raspberrypi」を検索した場合、検索結果はタイムアウト (300 秒以上) となった。

5. 考察

5.1 分散システムの設計

DB を利用したシステムと比べて提案手法は約 20 倍以上の性能となった。この理由として、ある種の検索に特化してプログラムを組んだこと、また、プログラム自体を C++で実装したことが挙げられる。検索に適したデータ構造でメモリ上に展開したことによって高速化されたと考えられる。また、C++でプログラムを作成してコンパイル時に最適化をかけたことで、インタープリタ方式と比べて高速に実行されたと考えられる。

また提案手法の中でも、複数の計算機に処理を分散させることで性能が十数倍に向上した。単一構成の場合と比べ、メモリ使用量や CPU、ファイル読み込みなど、計算機リソース面での有利さが現れた結果で

表 2. 各システムの実行時間

Tag	件数	提案手法 [s]		DB [s]	
		(複数構成)	(単一構成)	(Index なし)	(Index あり)
dog	9642	**0.666**	11.559	80.593	19.198
beach	30155	**1.299**	33.413	57.743	40.652
flower	14837	**0.964**	17.315	67.132	26.709
raspberrypi	138	0.703	**0.138**	(Time out: > 300)	1.832
sky	13135	**1.034**	14.770	66.015	30.595

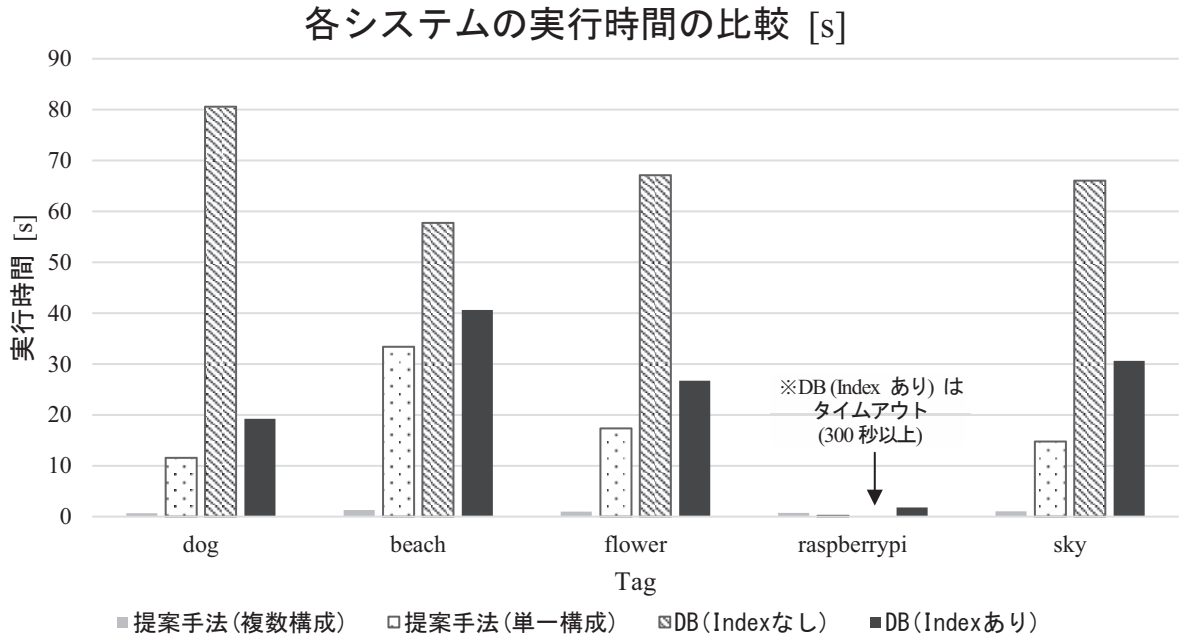

図 8. 各システムの実行時間の比較

あると言える。その一方「raspberrypi」で検索した場合など、検索時の計算量が少ない場合、単一構成の方が高速だった。複数構成の方が遅かった理由は、実行時間の内、各計算機へのタスクの割り振りやネットワーク通信のオーバーヘッドによるものと考えられる。単一構成で実行する際はタスクを割り振る必要もなく、またネットワーク上の通信も介さない為、複数構成よりも有利であったと考えられる。

これらの実験結果から、同じ検索方法でも、複数の計算機に分散処理させることで高速化できる可能性を示すことができた。ただしそのためには、ネットワーク上の通信はできる限り少なくした上で、各計算機の計算機リソース (メモリ、CPU、IO) の性能を引き出せるようにアルゴリズム・データ構造・ハードウェア構成を設計することが重要であると考えられる。

また実運用を考えたとき、今回の実装は並列性にのみ着目しているが、分散システムとして運用するには、規模透過性も考える必要がある。今回実装したシステムでは、サーバ機の数を変更する際はプログラムを一部変更し、再コンパイルしなければならず、これではサーバ機数を変更する場合の変更が容易ではない。システムが大規模になっても、プログラムやシステムの仕様に変更を加えることなく対応できるような仕組みを組み込んでおく必要がある。例えば、サーバ IP とデータの管理範囲をリスト化したファイルを用意する方法や、お互いのデータ管理範囲をネットワーク上で共有し、自律的に管理範囲を決定する仕組みを実装する方法などによって、規模透過性を上げることができると考えられる。また可用性も考慮し、冗長構成を検討しておく必要もある。一部の計算機に不具合が発生しても別の計算機でそれをカバーする工夫も必要となる。

さらに情報セキュリティ 3 要素のうち、完全性・機密性も考慮すると更に頑健なシステムとなると思われる。冗長構成をとったシステム上でデータが更新された際、計算機間でデータの整合性をとる必要がある。この時にサーバ間でデータの不整合が生じると、計算結果に矛盾が発生し、完全性が損なわれることになる。従ってその対策も必要と考えられる。また、分散システムではデータを複数の計算機に分散させることになるため、各計算機自体のセキュリティ対策が求められるほか、計算機を繋ぐネットワーク自体のセキュリティも考慮する必要があると考えられる。

5.2 開発環境について

今回の開発では、バージョン管理ツールを介して開発環境と本番環境でプログラムを管理するようにした。これによって、テスト環境で細かくテストできるだけでなく、迅速に各計算機に展開できるメリットが生まれ、開発効率は向上した。一方、これはクロス開発の弊害ともいえるが、開発環境と本番環境のアーキテクチャの差によって、開発時には想定していないバグが混入する可能性がある。実際今回は、開発環境である仮想環境は 64 bit OS を利用し、一方で本番環境は 32 bit OS を利用したが、開発中、これが原因となるバグに遭遇した。ID を格納するために当初 long 型を利用していたが、C++ における long 型のバイト長は開発環境上で 8 B であるのに対し、本番環境では 4 B となっていた。ID を格納するためには 8 B 必要だったため、本番環境では想定外の挙動となった。このバグの対応として、両環境でともに 8 B となる long long 型を採用することで解決した。このように、クロス開発の際はアーキテクチャの違いを踏まえて、注意してデータ型を決定する必要がある。また、本番環境に近いアーキテクチャを開発環境として選択するのも対策として有効と思われる。

6. まとめ・今後の展望

今回の実験を通じて、サーバプログラムを組む場合に、ハードウェア・ソフトウェアの限界を考慮して設計することの重要性を改めて認識した。特に想定した性能が出ない場合は、データ構造は適切であるか、通信の内容や手順に無駄がないか、アルゴリズムは適切か、計算機リソースの過不足はないか等、チューニングすべきポイントを見直すことで、計算機性能を十分に引き出すことができるだろう。

またバージョン管理ツールを利用したクロス開発は、分散システムに限らず一般のプロジェクトにも適用可能である。今後、業務で情報システムを開発する際は、積極的に今回の方法を採用していきたい。

謝辞

学生実験の内容を実施するにあたりまして、多くのご指導・アドバイスをいただきました、技術部情報部門 水野匠 様、藤戸翔 様、柴田頼紀 様に感謝申し上げます。また、ネットワークやデータベース等の技術・知識をご教授いただきました、静岡大学情報学の先生方に感謝申し上げます。

引用文献

[1] アンドリュー・S・タネンバウム, マールテン・ファン・スティーン著; 水野忠則, 宮西洋太郎, 鈴木健二, 西山智, 佐藤文明, 東野輝夫 訳 :「分散システム 原理とパラダイム」ピアソン・エデュケーション (2003), p.1-p.2

[2] Coulouris, George, Jean Dollimore, Tim Kindberg 著; 水野忠則, 福岡久雄, 斎藤雅史, 山口義一 訳 :「分散システム コンセプトとデザイン 第二版」電気書院 (1997), p.27-p.37

[3] Raspberry Pi :「Raspberry Pi 3 Model B」, < https://www.raspberrypi.com/products/raspberry-pi-3-model-b/> (2022 年 11 月 4 日データ取得)