平成 24 年度卒業論文

論文題目

群知能のクラウドファイルシステムへの応用

神奈川大学 工学部 電子情報フロンティア学科
学籍番号　200902789
道下　裕介

指導担当者　木下宏揚　教授

目 次

図 目 次

第1章　序論

1.1　背景

近年のコンピュータの演算性能の向上・記憶装置の大容量化により、画像や映像が高精細化したり、大量の情報からの検索速度が飛躍的に向上するなどして、私たちがコンピュータで扱うデータの量は著しく増大した。それに加えて通信ネットワークの帯域幅の拡大によって、クラウドコンピューティングが急速に普及し、ネットワークを経由して記録媒体を持ち歩くことなく大量のデータにアクセスすることが可能になっている。また、データだけでなくコンピュータの機能そのものさえにもネットワークを経由してアクセスすることも可能となり、私たちはもはやデータの入出力を受け持つ端末さえ持っていれば十分な程である。

1.2　問題点と解決策

このような状況の影で、ネットワーク上に存在し流通する情報の量は爆発的に増大している。それにより、例えばクラウドコンピューティングにおけるストレージでは、従来の木構造によるファイルシステムでは階層構造が複雑化し、使用者が階層構造を逐一確認してファイルを管理するには複雑すぎる構造となって、ファイルの参照が難しくなると考えられる。また、Web 検索に代表されるようなキーワードによる検索においては、検索元ファイルの数が増えることにより、検索結果の候補数も増え、さらなる絞り込みが必要となり所要時間が長くなったり、絞り込むことによって、価値があっても使われない情報が増えることが予想される。

そこで、それらの問題の解決を図るため、クラウド上のファイルが群れを形成することにより自動的に整理・分類され、関連するファイルを見やすく使用者に提示するファイルシステムを作ることを考えた。ここでは、このファイルシステムをクラウドファイルシステムと呼ぶ。

クラウドファイルシステムでは、データの増大による複雑なファイル管理や関連ファイルの抽出は、ファイルが群れを作ることにより自動的に行われ、使用者の手間を必要としない。また、使用者による定義や指定されたキーワードによる抽出とは異なる、ファイルの持つ特性により相互に定義される新たな関係性により、使用者に情報を提示し新たな視点を与えることが可能となる。

1.3 研究経緯

ファイルが群れを作る上で、従来のファイルシステムでは階層的構造とファイルの識別情報しかなく、ファイルの群れ、群れをなすためのファイルの振る舞いを記述するという機能は組み込まれていない。私たちは、クラウドファイルシステムに従来のファイルシステムに加えて、群知能モデルを用いてファイルごとに振る舞いを定義し、ファイルが相互の関係性を基に類似・関連するファイル同士で群れを作ることにより、ファイルを自動的に整理・分類して使用者に提示する機能を付加することを目指している [1]。

群知能モデルの一例として、鳥の群れをコンピュータ上で再現するためのモデルである Boid モデル [2] が挙げられる。このモデルは、非常に単純な法則だけで鳥の群れを高い完成度で再現することの可能なモデルであるが、私たちはこのモデルをクラウドファイルシステムの群知能モデルの元にすることができないかと考えた。

1.4 研究内容

本来生物の群がる行動を再現するための Boid モデルをクラウドファイルシステムにおいて群知能モデルとして利用するにあたって、Boid モデルをアプリケーションとしてプログラミングし、クラウドファイルシステムに応用する上での改良を施した、新しい群知能モデルの開発を行う。

また、この群知能モデルを開発するにあたって、クラウドコンピューティングにおける主要な端末である、タブレットやスマートフォンに多く採用されている、Andoroid プラットフォーム上で動作するソフトウェアとして開発することにより、それらの端末との親和性、同様に Android プラットフォーム上で動作するクラウドコンピューティングに関係するソフトウェアとの連携を容易なものとし、利便性の向上を図る。

第2章　基礎知識

2.1　クラウド・コンピューティング

2.1.1　定義

アメリカ国立標準技術研究所 (NIST) によれば [3]、最小限の管理作業とサービスプロバイダとの対話により、迅速に供給および開放が可能な、ネットワーク・サーバ・ストレージ・アプリケーション・サービスを含む構成の自在なコンピューティングリソースに場所を問わず便利に、オンデマンドにアクセスすることを可能にするモデルをクラウドコンピューティングという。このモデルは5つの主要な特徴と3つのサービスモデル、4つの展開モデルによって構成される。

2.1.2　主要な特徴

- オンデマンドのセルフサービス

使用者は自動的に、サーバのリソースやネットワークストレージのようなコンピューティング能力の供給を一方的に受けることができ、各サービスプロバイダと対話する必要がない。

- 幅広いネットワーク・アクセス

コンピューティングリソースはネットワーク上から利用可能で、携帯電話・タブレット・ラップトップ・ワークステーションのような異なる種類のシン/シック・クライアントからの利用を促進する標準的なメカニズムを通してアクセスできる。

- リソースの蓄え

プロバイダのコンピューティングリソースはマルチテナントモデルを用いて、異なる物理リソース及び仮想リソースを、顧客の必要に応じて割り当てまたは再割り当てすることにより複数の顧客に提供されるために蓄えられる。一般的に顧客は提供されるリソースの場所を知らない、決めることができないという点で、独立性がありますが、国・州・データセンターなどの抽象度の高いレベルでは場所を指定できるかもしれない。これらのリソースの例としては、ストレージ・処理・メモリ・ネットワーク帯域などが含まれる。

- 高い柔軟性

コンピューティング能力はいくつかのケースでは自動的に、顧客の需要に応じて外側にも内側にも素早く調整するために、柔軟に供給また開放することができる。顧客は供給できる能力は無制限にあるとみなすことができ、いつでも任意の量を割り当てることができる。

- 最適化サービス

クラウドシステムはそれぞれのサービスに適切な抽象化レベルでの計測機能 (ストレージ・処理能力・帯域幅およびアクティブユーザー数) を活用することにより、自動的にリソースの使用を制御し最適化する。リソースの使用状況は監視・制御・報告され、プロバイダとサービスを利用している顧客の双方に透明性を提供する。

2.1.3 サービスモデル

- サービスとしてのクラウドソフトウェア（SaaS）

顧客に提供される機能は、クラウドインフラ上で実行されるプロバイダのアプリケーションである。アプリケーションは Web ベースの E メールに例えられる Web ブラウザやプログラムインターフェースとしてシンクライアントインターフェースを通して様々なクライアントデバイスからアクセス可能である。顧客は、限られたユーザー固有のアプリケーションの構成設定の管理・制御する可能性以外は、ネットワーク・サーバ・OS・ストレージまたは個々のアプリケーション機能をを含むクラウドインフラの基礎を管理また制御していない。

- サービスとしてのクラウドプラットフォーム（PaaS）

顧客に提供される機能は、クラウドインフラ上に展開された、顧客自身がプログラミング言語・ライブラリ・サービスまたはプロバイダによってサポートされているツールによって作成および取得したアプリケーションである。顧客は展開されたアプリケーションと、おそらくアプリケーションホスティング環境の構成設定を制御しているが、ネットワーク・サーバ・OS・ストレージまたは個々のアプリケーション機能をを含むクラウドインフラの基礎を管理また制御していない。

- サービスとしてのクラウドインフラ（IaaS）

顧客に提供される機能は、処理・ストレージ・ネットワーク、または顧客が OS とアプリケーションを含む任意のソフトウェアを展開および実行できる、根本的なコンピューティングリソースである。顧客は、展開されたアプリケーションとホストファイアウォールのような選択されたネットワークコンポーネントの限られたコントロール以外の、ネットワーク・サーバ・OS・ストレージまたは個々のアプリケーション機能をを含むクラウドインフラの基礎を管理また制御していない。

2.1.4 展開モデル

- プライベートクラウド

クラウドインフラは、ビジネスユニットのような複数の顧客によって構成される単一の組織が排他的に使用するために提供される。それは、組織・第三者またはそれらのいくつかの組み合わせによって所有・管理・運営され、構内または構外に存在する。

- コミュニティクラウド

クラウドインフラは、ミッション・セキュリティ要件・ポリシー・コンプライアンス上の懸念事項のような懸念を共有している組織内の顧客で構成される特定のコミュニティが排他的に使用するために提供される。それは、一つかそれ以上の組織・第三者またはそれらのいくつかの組み合わせによって所有・管理・運営され、構内または構外に存在する。

- パブリッククラウド

クラウドインフラは一般団体によりオープンに使用するために提供される。それは企業・教育機関・政府組織またはそれらの複数により所有・管理・運営され、クラウドプロバイダの敷地内に設置される。

- ハイブリットクラウド

クラウドインフラは固有のまま存在するが、データとアプリケーションの可搬性を実現した標準化された、もしくは独自技術 (例: クラウド間のロードバランスをとるためのクラウドバースト) により結ばれた 2 つ以上の異なるプライベート・コミュニティおよびパブリッククラウドインフラの複合体である。

2.2 Boid モデル

Boid モデルとは、元々鳥の群れを再現するためにアメリカの Craig Reynolds 氏 [2] により提案されたモデル [4, 5, 6, 7, 8, 9] で、以下の 3 つのルールにより定義される [10]。

分離 (Separation)
他者との衝突を避けるために一定の距離を取る

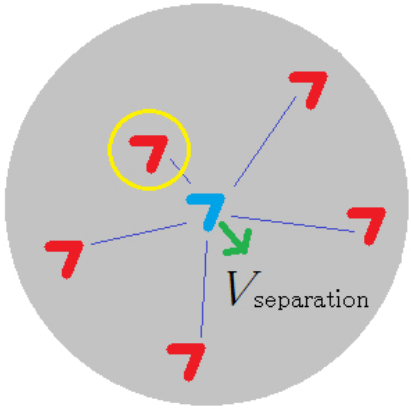

図 2.1: Separation

整列 (Alignment)
群れの進む方向に自らの進行方向を合わせる

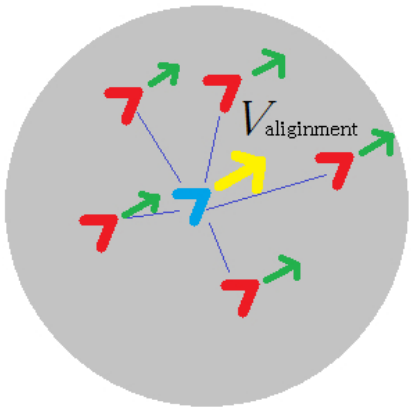

図 2.2: Alignment

結合 (Cohesion)
群れの中心に向かおうとする

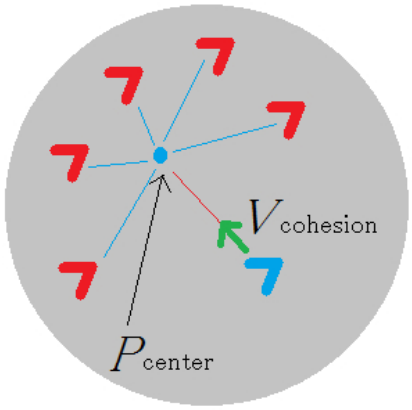

図 2.3: Cohesion

2.3 Android

Android とは、スマートフォン・タブレット PC などのモバイルデバイス向けのオープンでフリーなソフトウェアプラットフォームである。ここで言うソフトウェアプラットフォームとは、アプリケーションやアプリケーションフレームワークのみではなく OS やミドルウェアなども含んだ広範にわたるソフトウェアの集合体を指す [11, 12, 13]。このことから、Android ではソフトウェア開発は大きく 2 つの視点に分けて考えることができる。

- ある端末の上で動作する Android プラットフォームを作る

Android はソフトウェアの集合体であるため、最終的にはあるハードウェアの上に載せて動かす必要がある。Android とハードウェアを結び付けていく作業のことをポーティング（porting）と呼ぶ。この領域の開発を行うには、特に Linux カーネルやデバイスドライバに関する知識が必要不可欠。アプリケーションの開発者は、Android プラットフォームを直接操作することはなく、アプリケーションフレームワークの機能を通じて利用する形になる。

- Android プラットフォームの上で動作するアプリケーションを作る

開発者は「Android SDK」として提供されているアプリケーションフレームワークの機能や開発ツールを用いてアプリケーションを開発する。Android でアプリケーションを開発するためには、Java[14] やアプリケーションフレームワークに関する知識が必要になる。

2.3.1 Android のアーキテクチャ

アーキテクチャは大きく 4 つの層、5 つの領域に分かれている。

- Linux カーネル層

最下層となる「Linux カーネル」は、バージョン 2.6 を元にモバイルデバイス向けに変更が加えられている。システムサービスをはじめとしたアプリケーションを実行するために必要な基本的な機能を提供する、いわば基礎となる層。他の 3 つの層で動作する機能は、この Linux カーネル上で動作する。

- ライブラリ層

「Linux カーネル」層の上位層は、「ライブラリ」層となる。Android は、さまざまなコンポーネントから使用されるライブラリを提供する。ライブラリは、C や C++言語で作成されたライブラリを含む。これらの機能は、基本的にアプリケーション開発者が直接使用することはなく、上位層であるアプリケーションフレームワーク層の機能を通じて使用されることになる。

- Android ランタイム

「ライブラリ」層と同レベルの機能として「Android ランタイム」がある。Android ランタイムは Java 言語に準拠するコアライブラリと Android アプリケーションを実行する Dalvik 仮想マシンにより構成されている。

- アプリケーションフレームワーク層

「ライブラリ」層の上位層は、「アプリケーションフレームワーク」層になる。アプリケーション開発者は、SDK にあらかじめ含まれているアプリケーションで使用されているものと同じクラスを使用することができる。アプリケーションを構築する際に使用するアーキテクチャは、コンポーネントの再利用が簡単にできるように設計されている。

- アプリケーション層

最上位の層が「アプリケーション」層。SDK には、電話や Web ブラウザなどの実行可能なアプリケーションがあらかじめ含まれている。アプリケーション開発者が作成したさまざまなアプリケーションもこの層に位置づけられる。

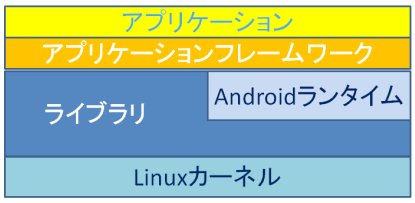

図 2.4: Android のアーキテクチャ

2.3.2 アプリケーション開発者の視点から見た Android の特徴

- Java 言語を使用してアプリケーションを作成する
- サーバアプリケーション開発で使用されているものに近いアプリケーションフレームワークが提供されている。

Web アプリケーションを構築する際によく使用される MVC2 パターンの要素（Model、View、Controller の 3 つの要素）を Android アプリケーションの構成要素に対応づけることができる。

- Java 言語と他の言語の使い分けをしている。

ライブラリ層には、モバイルデバイスに含まれるハードウェアを制御するためのソフトウェアが含まれている。この部分については C/C++言語などの Java 以外の言語で記述されている。Java 言語は、C/C++言語と比較した場合、実行速度やタイミング制御、メモリの管理などを不得意としている。なので、Android はそうした不得意な箇所には適切な言語で実装を行い、アプリケーション開発者には意識させないクラスのインターフェイスを提供している。

2.3.3 アプリケーション開発の流れ

開発作業の流れは、1) プロジェクトの作成 2) ソースコードの作成 3) アプリケーションの実行 の 3 つに分けることができる。

1) プロジェクトの作成 ••• プロジェクトはアプリケーションの単位で作成する。
2) ソースコードの作成
3) アプリケーションの実行 ••• アプリケーションが完成したら、Android エミュレータ上でアプリケーションを実行する。その手順は以下の通りである。

1. Java プログラムのコンパイル
2. Java バイトコード（.class）を Android バイトコード（.dex）に変換
3. Android アプリケーションをパッケージング（.apk ファイルの作成）

4. Android エミュレータの起動
5. アプリケーションを Android エミュレータにインストール
6. アプリケーションを Android エミュレータ上で起動

上記の作業が正常に完了したら、Android エミュレータ上でアプリケーションが動作する。

2.3.4 アプリケーションを構成する要素

Android では、クラスとして提供されるコンポーネント要素（API）を開発者が組み合わせることで 1 つのアプリケーションを作成する。このクラスには 2 つのタイプがある。

- Android アプリケーションとして動作するために必要なクラス
- Android アプリケーションの機能を提供するクラス

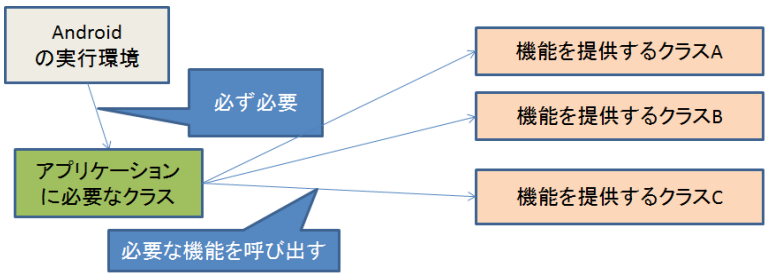

図 2.5: Android アプリケーションを構成するクラスと 2 つのクラスの関係性

Android の実行環境は、システム上にアプリケーションが必要となるクラスに紐づけたプロセスを作り、クラスの呼び出しを通じてアプリケーションの実行を制御する。この特別な役割を持ったクラスには 4 つの要素がある。

- アクティビティ

ユーザーとアプリケーションとの間で行われるやり取り全般を仲介する。

- インテント

アプリケーションの実行時に各要素を紐づける

- サービス

ユーザーの画面に対する操作に依存することなく処理を実行する。(バックグラウンド)

- コンテンツプロバイダ

アプリケーション間で情報を共有する。

アプリケーションの開発において、これらの要素が一度にすべて必要になるわけではない。たとえば、ユーザーが画面に対して操作を行うような標準的なモバイルアプリケーションでは、アクティビティを利用してアプリケーションを構築する。また、画面上で行う必要のない処理、いわゆるバックグラウンド処理と呼ばれるようなタイプのアプリケーションでは、サービスのみを使用するといった具合である。

Android アプリケーションとして機能を提供するためのクラスには、モバイルデバイスに装備されたハードウェアを利用するためのクラスを含む。また、アプリケーションを作成するための基本的な機能、たとえば画面制御やデータ制御などの機能も提供する。さらには、Java 言語のレベルで提供されるクラスも広い意味ではこのタイプに含めてよい。

第3章　群知能モデルの改良点

従来のファイルシステムにファイルの振る舞いを記述し、振る舞いから群れを作り保存する機能を付加しクラウドファイルシステムとするために、ファイルの振る舞いを定義しファイルが群れを作るための群知能モデルを提案する。鳥の“群れ”を再現するBoidモデルをベースに、ファイルの群れを定義する群知能モデルを開発していく。

その開発行程の一環として、Boidモデルプログラム[15]をAndroid上で動作するアプリケーションとして記述し、クラウドファイルシステムで利用するために次の機能を加えていくことによって、クラウドファイルシステムの群知能モデルの元となるアプリケーションとする。

3.1　ファイルの属性による分類を表現

Boidモデルでは基本的にある単一の種の鳥の群れを表現する。つまり、エージェントは皆互いに仲間であり、他のどのエージェントとも群れを作ろうとする。しかしクラウドシステム上のデータは各々異なっており、ファイルが似た属性とのみ群れを形成するような群知能モデルでなければ、データを整理・分類することはできない。

そこで、Boidモデル上の各エージェントに異なる属性を記録できるようにし、その属性に応じた群れを形成するような群知能モデルにする必要がある。実際のデータにおいては、データの形式・ファイル名・作成/更新/閲覧日時・サイズ・キーワード・使用者が付けたタグ等、様々なものがあるが、今回は色をそれらに見立ててBoidモデルへこの機能を追加する。これによって、Boidモデルにエージェントを分類する機能が付加され、クラウドファイルシステムにおいてファイルを整理・分類する働きが期待できる。

群知能モデルにおいて、この動きはBoidモデルの特性のうち群れを作るのに大きく関係するAlignmentおよびCohesionにおいて色を識別して同種のエージェントにのみ機能させることにより実現させる。

3.2 視認性向上のための群れの安定化

自然界の群れをなす鳥が飛び続けているように、Boid モデルにおいてもエージェントは絶えず動き続けている。しかし、ユーザーがクラウドシステム上のデータを群れとして見る際に、群れ、つまりファイル同士の関係性が常に動き続け、変化していては関係性を把握することが非常に難しい。

そこで、周辺の他エージェントの数により自身の群れの規模を認識して群れの完成度を定義し、エージェントの速度を調整する。群れの規模がある程度大きければ、その群れは情報の群れとして完成度が高く、ユーザーに把握しやすいよう移動速度を下げるのがふさわしい。一方、群れの規模が小さい、もしくは群れにエージェント自身しかいない場合、そのエージェントは情報の群れを形成しているとは言い難く、どこかの群れに属するために他エージェントを探して活発に動く必要があり、移動速度を上げるのが望ましい。これによって、群れの形成を極端に阻害することなく、使用者がファイルの群れを視覚的に見た際に、ファイルの関係性を用意に素早く把握することが可能となる。

この特性は、Boid モデルの 3 つの特性に加えて、一定距離内の周辺の他エージェントの数により自身のエージェントの最大速度を調節することにより実現させる。

第4章　群知能モデルのアプリケーション開発

4.1　開発概要

Android 端末上で動作する Boid モデルをもとに、クラウドファイルシステムの基礎となるファイルの群れを定義する群知能モデルを開発する。

4.1.1　開発環境

- JRE (Java Runtime Environment) Version 7 Update 13
- JDK (java Development Kit) Version 7 Update 13
- Eclipse 4.2.0 Juno
- Android SDK (Software development Kit) 21.0.1
- ADT (Android Development Tools) Plugin for Eclipse 21.0.1
- Andoroid 端末: SONY Xperia VL SOL21

OS	Android 4.0.4
CPU	Qualcomm Snapdragon S4 MSM8960 1.5GHz(デュアルコア)
RAM	1GB
ROM	16GB
メインディスプレイ	4.3 インチ 1280x720pixel 342ppi

表 4.1: Android 端末の主な仕様

4.2 開発手順

4.2.1 開発環境のセットアップ

開発に必要なソフトウェア類をインストールする。これらは全て無料で公開されており、公開元からダウンロードすることができる。

JDK(Java Development Kit)
Java 開発キット。Java プログラムを開発するために必要なプログラムが含まれている。

JRE(Java Runtime Environment)
Java ランタイム環境。Java を実行するためのソフトウェア。JDK とセットでインストールされる。

Eclipse
オープンソースの統合開発環境。プラグインと組み合わせることにより Java をはじめとする様々な言語のプログラム開発環境を提供する。

Android SDK
Android ソフトウェア開発キット。Android 上で動作するアプリケーションを開発するために必要なソフトウェアである。

ADT Plugin for Eclipse
Eclipse で Android のアプリケーションを開発するためのプラグインソフトウェアである。Eclipse 上から追加ソフトウェアの形でインストールを行う。

4.2.2 AVD(Android Virtual Device) の作成と起動

開発したアプリケーションのデバッグを行うための仮想の Android デバイスを PC 上に作成しておく。図 4.1 の画面において、使用する Andoroid OS のバージョン・画面解像度・外部メモリの容量等を設定する。

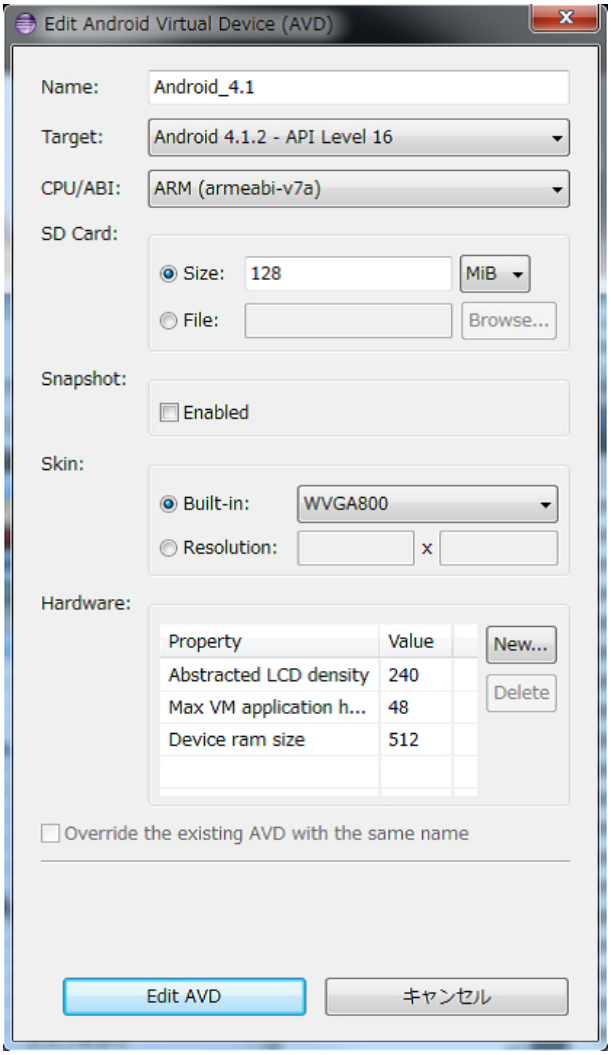

図 4.1: AVD の設定画面

4.2.3 プロジェクトの作成

Eclipse で新規プロジェクトを作成し、プログラムを記述する。

4.3 作成したアプリケーションの詳細

4.3.1 ユーザー定義クラス

以下に主要なユーザー定義クラスの機能を簡潔に述べる。

CloudFileSystem

Activity と呼ばれる Android のアプリケーションの基礎となるクラスを継承した基本クラス。アプリケーションを起動すると最初に実行される部分である。

mainSurfaceView

SurfaceView[16] と呼ばれる Andoroid において比較的高速な描画をするためのクラスから継承した、このプログラムにおいて画面の描画を行うためのクラスである。

run

このプログラムのメイン処理スレッドで、このクラスをループさせることで再帰的にエージェントの動きを定義する。

onTouchEvent

ユーザーが端末の画面を触れた際に実行されるクラス。ファイルを想定したエージェントを生成し、初期動作を定義する。

doDraw

図形を描画するクラス。各エージェントの座標を取得し、指定された色の丸印でエージェントの位置を描画する。

4.3.2 汎用クラス

- getDist

i,j 番目のエージェント同士の距離を返す。

```
private float getDist(int i, int j){
 return (float)Math.sqrt(Math.pow(xPos.get(i) - xPos.get(j),2) + Math.pow(yPos.get(i) - yPos.get(j), 2));
}
```

- colorCheck

i,j 番目のエージェントの色が一致しているかを検証する。一致していれば真を、していなければ否を返す。

```
 private boolean colorCheck(int i, int j){
   float colorDiff = 0, rDiff, gDiff, bDiff;

   rDiff = Math.abs(red.get(j) - red.get(i));
   gDiff = Math.abs(gre.get(j) - gre.get(i));
   bDiff = Math.abs(blu.get(j) - blu.get(i));
   colorDiff = (rDiff + gDiff + bDiff);

   if (colorDiff == 0)return true;
   else return false;
 }
```

4.3.3 群知能モデルの記述

Boid モデルの 3 つの特性を、クラウドファイルシステムを想定し改良し表したクラスの機能を述べる。

- Separation

自身以外のエージェントとの距離が 30 以下であればエージェントの方向と逆向きに加速する。この特性は単にエージェント同士の衝突を避けるためのものなので、色による区別は行わない。

```
private void Separation(){
 float dirx, diry;

 for(j = 0; j < n; j++){
  if (i == j);
  else if (getDist(i, j) < 30){
   dirx = xPos.get(j) - xPos.get(i);
   diry = yPos.get(j) - yPos.get(i);
   dx.set(i, dx.get(i) - dirx / 20);
   dy.set(i, dy.get(i) - diry / 20);
  }
 }
}
```

- Alignment

第 3.1 節で取り上げた属性ごとの群れを表現するために、自身以外の距離が 60 以内、かつ色が一致するエージェントの進行方向の平均を自身の進行方向に加える。

```
private void Alignment(){
 float avx = 0, avy = 0;
 int count = 0;

 for(j = 0; j < n; j++){
  if (getDist(i, j) == 0);
  else if (getDist(i, j) < 60 && colorCheck(i, j)){
   avx += dx.get(j);
   avy += dy.get(j);
   count++;
  }
 }
 if (count != 0){
  avx /= count;
  avy /= count;
  dx.set(i, dx.get(i) + avx / 100);
  dy.set(i, dy.get(i) + avy / 100);
 }
}
```

- Cohesion

第 3.1 節で取り上げた属性ごとの群れを表現するために、自身以外の距離が 100 以内、かつ色が一致するエージェントの座標の平均に向かって加速する。

```
private void Cohesion(){
 float gx = 0, gy = 0;
 float cx, cy;
 int count = 0;

 for(j = 0; j < n; j++){
  if (i == j);
  else if (getDist(i, j) < 100 && colorCheck(i, j)){
   gx += xPos.get(j);
   gy += yPos.get(j);
   count++;
  }
 }
 if (count != 0){
  gx /= count;
  gy /= count;
  cx = gx - xPos.get(i);
  cy = gy - yPos.get(i);
  dx.set(i, dx.get(i) + cx / 50);
  dy.set(i, dy.get(i) + cy / 50);
 }
}
```

- grStability

第 3.2 節で取り上げた、群れの関係性の把握を用意にするための手段として、群れの規模により属するエージェントの移動速度を変化させる。

自身から距離が 100 以内のエージェントの個数を最大 20 個の範囲内でカウントし、より多いほど自身の移動速度を落とす。

```
private void grStability(){
 int count = 0;

 for(j = 0; j < n; j++){
  if (i == j);
  else if (getDist(i, j) < 100){
   count++;
  }
 }
 if (count > 21)count = 20;
  dxmax.set(i, 3f - count / 10);
  dymax.set(i, 3f - count / 10);
}
```

4.3.4 実行結果

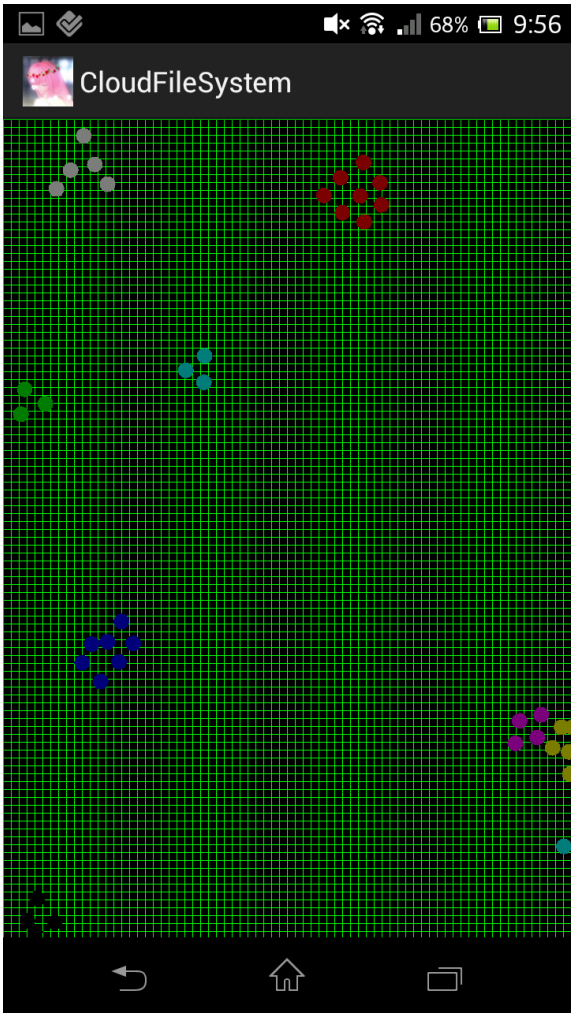

図 4.2: 開発したアプリケーションの実行画面

図 4.2 にアプリケーションの実行画面を表す。紙面では動きを表すことはできないが、同じ色のエージェント同士で各々群れを形成していることがわかる。同じ色同士ではエージェントを集める整列・結合の特性が働き、異なる色の間では分離特性のみが機能することにより、エージェントの属性ごとに群れを定義することが可能となった。

また、プログラム上では群れの数が増加した際のエージェントの速度低下が確認できた。

第5章　結論

5.1　開発のまとめ

クラウドファイルシステムで用いるBoidモデルをもとにした群知能モデルを開発するにあたって、Andoroidプラットフォーム上でBoidモデルをプログラミングし、クラウドファイルシステムに用いる上での改良点として、エージェントの色ごとに群れを作る機能と、群れのエージェントの数に応じて移動速度を調節させる機能を追加することができた。エージェントは色ごとに集まって速度を落とし、画面上に色ごとの分類として表示された。この動きはファイルを整理・分類する上で便利であり、この群知能モデルをもとに、実際にファイルから得られる属性を取得してモデルに反映させることによって、クラウド上のファイルで群れを作り、その振る舞いを記述することで、クラウドファイルシステムに役立てることができると期待している。

5.2　今後の課題

群知能モデルの原型は開発することができたが、実際にデータの集合に当てはめる際に振る舞いの元として用いるデータの要素の選定や、実際の振る舞いは今回の開発のように色を要素とした場合とは異なるため、振る舞いの検証とパラメータの調整、要素の組み合わせ方などを定めることが今後の課題である。

第6章　謝辞

本研究を行なうにあたり、終始熱心に御指導していただいた木下宏揚教授に心から感謝致します。また、様々な面で数多くの有益な御助言をしていただいた東洋ネットワークシステムズ株式会社の森住哲也氏に深く感謝致します。さらに、研究活動一般に様々な助言をいただきました宮田女史をはじめ、公私にわたり良き研究生活を送らせていただいた木下研究室の方々に感謝致します。

2013年　2月
道下　裕介

参考文献

[1] 磯村淳、木下宏揚“クラウドファイルシステム”
神奈川大学卒業論文 (2011)

[2] Craig Reynolds
“Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)”
SIGGRAPH ’87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques pp.25-34 (1987)

[3] Peter Mell / Timothy Grance “The NIST Definition of Cloud Computing”
NIST Special Publication 800-145
http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[4] Delgado-Mata C, Ibanez J, Bee S, et al.
“On the use of Virtual Animals with Artificial Fear in Virtual Environments”
New Generation Computing 25 (2): pp.145—169 (2007)

[5] Hartman C, Benes B
“Autonomous boids”
Computer Animation and Virtual Worlds 17 (3-4): pp.199—206 (2006)

[6] Lebar Bajec, I. and F. H. Heppner
“Organized flight in birds” (2009)

[7] Ibanez J, Gomez-Skarmeta A F, Blat J
“DJ-boids: emergent collective behavior as multichannel radio station programming”
Proceedings of the 8th international conference on Intelligent User Interfaces. pp.248-250 (2003)

[8] Moere A V
“Time-Varying Data Visualization Using Information Flocking Boids”
Proceedings of the IEEE Symposium on Information Visualization. pp.97-104 (2004)

[9] Cui Z, Shi Z
“Boid particle swarm optimisation”

International Journal of Innovative Computing and Applications 2 (2): pp.77—85 (2009)

[10] 安田 浩志、黒崎　寛、樽井 健人、中村 裕樹、滑川　技　“ Boid とは ” http://members.jcom.home.ne.jp/ibot/boid.html

[11] 江川 崇、竹端 進、山田 暁通、麻野 耕一、山岡 敏夫、藤井 大助、藤田 泰介、佐野 徹郎 :“ Google Android プログラミング入門 ” 株式会社豆蔵 (2009)

[12] 蓬田宏樹、他 “ Android の野望 ” 日経エレクトロニクス 2007 年 12 月 17 日号

[13] 服部武、藤岡雅宣 “ HSPA+/LTE/SAE 教科書 ” インプレス R & D(2009)

[14] 岩谷　宏 :“ JAVA の哲学 ” ソフトバンクパブリッシング (2001)

[15] “ boids 理論でデモを作ってみる ” http://n-yagi.0r2.net/as3/programming/boid/

[16] “ 描画用スレッドを実装できる SurfaceView の利用方法 ” Keisuke Oyama http://android.keicode.com/basics/surfaceview-1.php

付録

作成したAndroidアプリケーションのJavaソースコードを以下に示す。

[CloudFileSystem.java]

```
package aya.cfs;

import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.MotionEvent;
import java.util.ArrayList;
import java.util.Random;
import java.lang.Math;

public class CloudFileSystem extends Activity {

 @Override
  public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   setContentView(new mainSurfaceView(this));
 }
}

class mainSurfaceView extends SurfaceView implements SurfaceHolder.Callback,Runnable{
 Thread thread;
 boolean running;

 private int n = 0;
 private int i,j;

 ArrayList<Float> xPos = new ArrayList<Float>();
 ArrayList<Float> yPos = new ArrayList<Float>();

 ArrayList<Float> rad = new ArrayList<Float>();
 ArrayList<Float> dx = new ArrayList<Float>();
 ArrayList<Float> dy = new ArrayList<Float>();

 ArrayList<Float> dxmax = new ArrayList<Float>();
 ArrayList<Float> dymax = new ArrayList<Float>();

 ArrayList<Integer> red =new ArrayList<Integer>();
 ArrayList<Integer> gre =new ArrayList<Integer>();
 ArrayList<Integer> blu =new ArrayList<Integer>();

 private float scrWidth, scrHeight;
 private float xGrid, yGrid;

 public mainSurfaceView(Context context){
```

```
 super(context);

 getHolder().addCallback(this);
}

@Override
public void surfaceChanged (SurfaceHolder holder, int format, int width, int height){
 scrWidth = width;
 scrHeight = height;
}

@Override
public void surfaceCreated(SurfaceHolder holder){
 running = true;
 thread = new Thread(this);
 thread.start();
}

@Override
public void surfaceDestroyed(SurfaceHolder holder){
 running = false;
 while (thread.isAlive());
}

@Override
 public void run(){
 while(running){
  Log.v("MainActivity","run");
  for(i = 0; i < n; i++ ){
   Separation();
   Alignment();
   Cohesion();

   colorSeparation();
   rStability();

   if (dx.get(i) > dxmax.get(i))dx.set(i, dxmax.get(i));
   if (dx.get(i) < dxmax.get(i) * -1)dx.set(i, dxmax.get(i) * -1);
   if (dy.get(i) > dymax.get(i))dy.set(i, dymax.get(i));
   if (dy.get(i) < dymax.get(i) * -1)dy.set(i, dymax.get(i) * -1);

   if (xPos.get(i) < 0){
    dx.set(i, dx.get(i) * -1);
    xPos.set(i, 0f);
   }
   if (scrWidth < xPos.get(i)){
    dx.set(i, dx.get(i) * -1);
    xPos.set(i, scrWidth);
   }

   if (yPos.get(i) < 0){
    dy.set(i, dy.get(i) * -1);
    yPos.set(i, 0f);

   if (scrHeight < yPos.get(i)){
    dy.set(i, dy.get(i) * -1);
    yPos.set(i, scrHeight);
   }

   xPos.set(i, xPos.get(i) + dx.get(i));
   yPos.set(i, yPos.get(i) + dy.get(i));
  }
  doDraw(getHolder());
 }
}

@Override
```

```
public boolean onTouchEvent(MotionEvent touch){
 Random rnd = new Random();
 red.add(127 * rnd.nextInt(2));
 gre.add(127 * rnd.nextInt(2));
 blu.add(127 * rnd.nextInt(2));
 xPos.add(touch.getX());
 yPos.add(touch.getY());

 rad.add((float)(Math.random()*2*Math.PI));
 dx.add((float)Math.cos(rad.get(n))*5);
 dy.add((float)Math.sin(rad.get(n))*5);

 dxmax.add(3f);
 dymax.add(3f);
 n++;

 return super.onTouchEvent(touch);
}

private void doDraw(SurfaceHolder holder){
 Canvas canvas = holder.lockCanvas();

 Paint paint = new Paint();

 canvas.drawColor(Color.BLACK);
 paint.setColor(Color.GREEN);
 for(xGrid = 0; xGrid < scrWidth; xGrid += 10){
  canvas.drawLine(xGrid, 0, xGrid, scrHeight, paint);
 }
 for(yGrid = 0; yGrid < scrHeight; yGrid +=10){
  canvas.drawLine(0, yGrid, scrWidth, yGrid, paint);
 }

 for(i = 0; i < n; i++){
  paint.setColor(Color.rgb(red.get(i), gre.get(i), blu.get(i)));
  canvas.drawCircle(xPos.get(i), yPos.get(i), 10, paint);
 }
 holder.unlockCanvasAndPost(canvas);
}

private void Separation(){
 float dirx, diry;

 for(j = 0; j < n; j++){
  if (i == j);
  else if (getDist(i, j) < 30){
   dirx = xPos.get(j) - xPos.get(i);
   diry = yPos.get(j) - yPos.get(i);
   dx.set(i, dx.get(i) - dirx / 20);
   dy.set(i, dy.get(i) - diry / 20);
  }
 }
}

private void Alignment(){
 float avx = 0, avy = 0;
 int count = 0;

 for(j = 0; j < n; j++){
  if (getDist(i, j) == 0);
  else if (getDist(i, j) < 60 && colorCheck(i, j)){
   avx += dx.get(j);
   avy += dy.get(j);
   count++;
  }
 }
 if (count != 0){
```

```
   avx /= count;
   avy /= count;
   dx.set(i, dx.get(i) + avx / 100);
   dy.set(i, dy.get(i) + avy / 100);
  }
 }

 private void Cohesion(){
  float gx = 0, gy = 0;
  float cx, cy;
  int count = 0;

  for(j = 0; j < n; j++){
   if (i == j);
   else if (getDist(i, j) < 100 && colorCheck(i, j)){
    gx += xPos.get(j);
    gy += yPos.get(j);
    count++;
   }
  }
  if (count != 0){
   gx /= count;
   gy /= count;
   cx = gx - xPos.get(i);
   cy = gy - yPos.get(i);
   dx.set(i, dx.get(i) + cx / 50);
   dy.set(i, dy.get(i) + cy / 50);
  }
 }

   private boolean colorCheck(int i, int j){
     float colorDiff = 0, rDiff, gDiff, bDiff;

     rDiff = Math.abs(red.get(j) - red.get(i));
     gDiff = Math.abs(gre.get(j) - gre.get(i));
     bDiff = Math.abs(blu.get(j) - blu.get(i));
     colorDiff = (rDiff + gDiff + bDiff);

     if (colorDiff == 0)return true;
     else return false;
  }

 private void grStability(){
  int count = 0;

  for(j = 0; j < n; j++){
   if (i == j);
   else if (getDist(i, j) < 100){
    count++;
   }
  }
  if (count > 21)count = 20;
   dxmax.set(i, 3f - count / 10);
   dymax.set(i, 3f - count / 10);
 }

 private float getDist(int i, int j){
  return (float)Math.sqrt(Math.pow(xPos.get(i) - xPos.get(j),2) + Math.pow(yPos.get(i) - yPos.get(j), 2));
 }
}
```