複数のシミュレータを統合する大規模分散シミュレーションカーネル

小藤 哲彦　竹内 郁雄
電気通信大学

災害に対する最先端技術の適用と研究はきわめて緊急性が高い。筆者らは災害被害の最小化を目指す RoboCup-Rescue プロジェクトに参加し、大規模災害シミュレーションシステムの開発に携わっている。大規模災害シミュレーションのシステムには、(1) サブシミュレータの個別の開発やシミュレーションしたい災害に応じたシステム構成を実現するための、サブシミュレータのプラグイン機能、(2) 大規模な災害をシミュレーションする際に、計算機の台数を増やすことによって計算時間の増加を抑えるスケーラビリティが必要である。本研究では、これらの条件を満たすアーキテクチャおよびカーネルを開発し、簡単なシミュレーションを実行させることによって、その効果を測定した。

A Large-scale Distributed Simulation Kernel that Integrates Multiple Individual Simulators

Tetsuhiko Koto　Ikuo Takeuchi
The University of Electro-Communications

Integrated disaster simulation systems are important tools to decrease the damage from disasters. For integrated disaster simulation systems it is highly desirable to have the following features. (1) Sub-simulators can be plugged into the system. It allows us to develop sub-simulators individually, and to build a system from suitable sub-simulators. (2) For the sake of scalable disaster simulations, it can work on a cluster of variable number of computers so that disaster over larger area can be simulated without being slowed down. We developed a kernel of an integrated simulation system with these features. The paper describes the kernel and the architecture of the simulation system. It also describes the performance of the system.

1　背景

1995 年 1 月 17 日に発生した阪神・淡路大震災は 6,000 人を超える死者、100 万人を超える被災者を出し、我が国の防災・救命システムの不備を露呈した。日本は世界有数の地震頻発国であり、幾多の歴史的事実は必ず大災害が発生することを示している。地震だけではなく、台風による風水害、土砂崩れ、海難事故など、災害や事故は後を絶たない。さらに、宮城県沖、東海などでは近い将来にきわめて高い確率で大規模地震が発生すると予測されており、防災・救命救助システムに対する最先端技術の適用と研究はきわめて緊急性が高い。筆者らは、上記の視点で大規模災害の被害を最小化する情報処理技術の発展を目指す、RoboCup-Rescue プロジェクト [1] に参加し、災害シミュレータの開発に携わっている。そこで作成したプロトタイプシミュレータは、1.5km 四方の街における地震災害のシミュレーションを行うことができるが、シミュレーションのサイズに比例して、計算に時間がかかり、災害救助活動の実用に供するには能力が不足していた。この問題を解決するためには、シミュレーションシステムをスケーラブルなものにしなければならない。

2　特徴

本研究で開発したシミュレーションシステムは次の特徴を持つ。

分散環境での効率的な動作　大規模な災害のシミュレーションを可能にするために、シミュレーションの規模が大きくなったときに、シミュレーションの規模に比例して計算機を増やすことで、処理時間の増大を抑える。ただし、シミュレーション対象の世界において、地理的に離れた位置にあるオブジェクト間に直接の相互作用がない、あるいは相互作用の伝達に時間がかかることを前提にして成立している。

サブシミュレータのプラグイン　プロトタイプシミュレータと同様、地震、火災、交通など、現象に応じたサブシミュレータを個別に開発し、それらを統合してシミュレーションを行うことができる。用途に応じてサブシミュレータを差し替えたり、新しいドメインのサブシミュレータを追加することができる。

サブシミュレータに分散を意識させないアーキテクチャ　サブシミュレータは、自分が分散シミュレーションを行っていることを意識せずに分散シミュレーションを行うことができる。

スペースタイムベースのシミュレーション　スペースタイム [2] の概念を用いてシミュレーションシステムを抽象化した。シミュレーションシステムの目的は、縦軸がシミュレーション対象の世界の状態を表す変数、横軸がシミュレーション対象の世界における時間であるような表 (スペースタイムグラフ) を埋めることである。本研究のシステムでは、複数のサブシミュレータが 1 つのスペースタイムグラフを共有し、サブシミュレータ間のインタラクションは、スペースタイムグラフ上の値の読み書きによって実現される。

保守的アプローチ　保守的なシミュレーションシステムである。すなわち、一度決定されたシミュレーション結果が、計算の進行に伴って修正されることはない。つまり、スペースタイムグラフへ値を設定すると、その値は変更できない。

オブジェクトプロパティモデル　シミュレーション対象の世界は、プロパティをもつオブジェクトとしてモデル化される。プロパティやオブジェクトの種類は、動的に増やすことができる。災害シミュレーションであれば、建物や市民がオブジェクト、火の勢いやケガの程度がプロパティの例である。オブジェクトは動的に増減させることができる。

マルチプラットホーム　Java によって実装し、計算機間の通信に HORB[3] を用いたため、TCP/IP 接続された Linux/Windows の動作する PC を含む、様々な計算機上で動作させることができる。

オブジェクト指向デザイン　オブジェクト指向で記述されているため、1 つのインターフェイスに対して様々な実装を許すことができ、拡張性がある。

このシステムにおいて、カーネルは、サブシミュレータ間の通信を管理し、シミュレーションを進行させる役割を果たす。このシステムによって、世界中の研究者が開発したサブシミュレータをプラグインし、様々な現象が複雑に絡み合った大規模災害救助シミュレーションシステムを作成することができる。そして、そのシミュレーションシステムによって、災害時の被害予測、救助戦略の立案支援、災害に強い都市計画の支援、防災訓練時における仮想的な災害の提供、などの効果が期待できる。

3　アーキテクチャ

本研究では、まず分散を行わないシミュレーションシステム (非分散システム) を構築し、非分散システムを利用して、分散シミュレーションを行うシステム (分散システム) を構築した。

3.1　非分散システム

非分散システムは、カーネルとサブシミュレータ群からなる。カーネルは個々のサブシミュレータに対して、スペースタイムグラフを保持する共有メモリを仮想的に提供する。サブシミュレータがカーネルから提供された共有メモリに対して読み書きを行うことで、シミュレーションが進行する。広義には、シミュレーションを視覚化するモジュール、シミュレーションを記録するモジュール、シミュレーション対象の世界の初期値を提供するモジュールなども、サブシミュレータ群に含まれる。

3.1.1　非分散システムのインターフェイス

以下は、非分散システムの基本的なインターフェイスの Java に基づいた記述である。

```
// A Interface for the Initialization
interface Memory {
    Writable registerWriter(
            String propertyName,
            MemoryWriter writer);
    void registerReader(
            String propertyName,
            CalledBackMemoryReader reader);
    Readable registerReader(
            String propertyName,
            MemoryReader reader);
}

// 3 Interfaces for the Simulation Progress
interface Writable {
    void writeToMemory(...);
}
interface CalledBackMemoryReader {
    void written(...);
}
interface Readable {
    Object getValue(String objectID, Time time);
    Object getAvarageValue(
            String objectID,
            Time start,
            Time end);

    void discardBefore(Time time);
    UpperBoundary getFixedUpperBoundary();
    UpperBoundary waitFixed(Time time);
}
```

Memory は、共有メモリを表すインターフェイスであるが、Memory 自体はシステムの初期化時にしかアクセスされない。サブシミュレータは初期化時に、registerWriter, registerReader メソッドによって、Memory に自分自身を登録し、メモリへのアクセス権を得る。メソッドの引数には、アクセスするプロパティの名前を指定する。複数のプロパティにアクセスする場合、複数回メソッドを呼び出す必要がある。

registerWriter メソッドによって書き込みを登録すると、Writable が返される。サブシミュレータは、Writable のメソッドを呼び出すことによって、共有メモリに対する書き込みを行うことができる。

registerReader メソッドは 2 種類ある。1 つめのメソッドによって読み込みを登録すると、登録したプロパティに対して書き込みがある毎に、registerReader メソッドに渡した CalledBackMemoryReader オブジェクトに対して、メソッド呼び出しが行われる。メモリ変化の差分のみを受け取ることになるため、サブシミュレータによってはプログラムが書きにくくなる場合がある。

2 つめの registerReader メソッドは、Readable を返す。サブシミュレータは、Readable のメソッドを呼び出すことによって、共有メモリを読むことができる。また、discardBefore メソッドによって、もう読み込みを行わない共有メモリの領域を宣言することができ、その領域の値を記憶するために使われていた計算機のメモリを開放することができる。また、waitFixed メソッドによって、読み込みたい領域が他のサブシミュレータによって書き込まれるまで、サブシミュレータのスレッドをサスペンドすることができる。2 つめの registerReader は、1 つめの registerReader とは異なり、書き込みをメモリ上に保持するので、メモリ使用量が多い。また、若干のオーバーヘッドがある。

シミュレーション世界中のオブジェクトの増減は、org.robocup.rescue.System オブジェクトの org.robocup.rescue.addedObjects プロパティと org.robocup.rescue.removedObjectIDs プロパティに対する書き込みによって実現される。シミュレーション対象の世界にどのようなオブジェクトが存在しているかは、org.robocup.rescue.objectIDs プロパティに記録される。

3.1.2 非分散システムの実装

Writable への書き込みに対する CalledBackMemoryReader の呼び出しを、Observer パターン [4] によって実装した。また、Writable への書き込みを、Readable からアクセス可能なデータ構造で蓄えるようにした。このデータ構造は、1 つのプロパティに対して複数の registerReader が呼び出されたとき、Readable によって共有され、メモリ消費量を抑える。

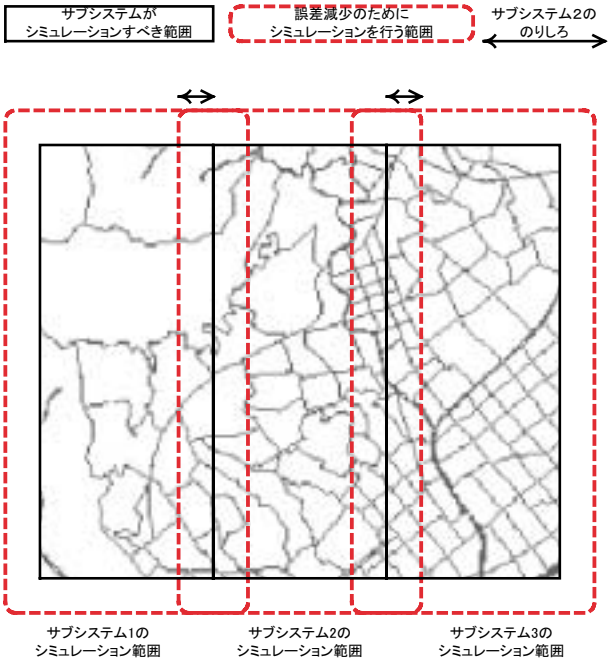

図 1: 分散システムのシミュレーション範囲

サブシミュレータの競合解消　複数のサブシミュレータが同一のプロパティに書き込みを行いたい場合がある。例えば、災害シミュレーションの場合、延焼シミュレータによって建物の火の勢いが設定されるだけでなく、着火シミュレータによっても火の勢いが設定される。このような場合に、異なるサブシミュレータからの書き込み競合を解消する仕組みを用意する必要がある。

本システムでは、マージャーと呼ばれる特殊なサブシミュレータを導入することで、この問題を解決した。複数のサブシミュレータが同一のプロパティに書き込む場合、内部的には、サブシミュレータごとに異なるプロパティが用意される。また、読み込み用にもう一つ異なるプロパティが用意される。マージャーは、書き込み用に用意されたプロパティを読み込んで、書き込まれた値をマージして、読み込み用に用意されたプロパティへ書き込みを行う。デフォルトでは、システム初期化時に先に登録されたサブシミュレータの書き込みを優先するマージャーが使用される。

サブシミュレータにとっては、自分が書き込みを行ったプロパティの値が、自分が書き込んだ値とは異なる場合があることになる。このため、サブシミュレータは、自分が書き込んだプロパティについて、書き込みの後読み込みを行い、その値を自分のシミュレーションモデルに反映させなければならない。

3.2 分散システム

分散システムは、複数の非分散システムを結合することで構築されるシミュレーションシステムである (図 1)。分散システム全体がシミュレーションする対象を、個々の非分散システムへ地理的に分割し、各非分散システムが行う小部分 (図 1 の「サブシステムがシミュレーションすべき範囲」) のシミュレーションの総体として、シミュレーションを行う。隣接する非分散システムの間では、シミュレートされる世界の時刻において定期的に、境界線から一定の範囲のシミュレーション結果を互いに教えあう。若干の語弊があるがこの教えあうことを、ここでは交換と呼ぶことにする。また、分散システムの要素としての非分散システムをサブシステムと呼ぶことにする。また、他のサブシステムからシミュレーション結果を教えられる範囲のことを (教えられる側のサブシステムにとっての) のりしろと呼ぶことにする (他のサブシステムへ教える範囲はのりしろに含まない)。

交換は、サブシステム間の通信による時間消費を抑えるために、シミュレーションの時間粒度に比べて長いインターバルで行われる。このため、サブシステムが交換によって知ったのりしろの情報と、隣接するサブシステムの行うシミュレーション結果との間の誤差は、時間がたつにつれて大きくなる。この誤差の影響を減らすため、サブシステムは、交換されたのりしろの部分も、自分がシミュレーションすべき範囲と同様に、シミュレーションの対象とする (図 1 の「誤差減少のためにシミュレーションを行う範囲」)。ただし、この拡張されたシミュレーション範囲は、分散システム全体のシミュレーション結果としては採用されない。

境界線付近では、隣接する 2 つのサブシステムは、ほぼ同じ入力に対して、同じアルゴリズムに基づいてシミュレーションを行うため、ほぼ同じ結果が得られることが期待できる。特に、オブジェクトが地理的に離れたオブジェクトに対して影響を与えるのに、距離に応じた時間がかかる場合、のりしろを十分大きく取り、交換のインターバルを十分小さくすれば、隣接するサブシステムの境界線付近のシミュレーション結果は、完全に一致する。

交換は、個々のサブシステムにとっては、シミュレーション中の動的なオブジェクトの増減としてあつかわれ、分散シミュレーションであることを意識する必要はない。交換の際に、のりしろ部分のオブジェクトを全て削除し、交換によってもたらされた情報を、オブジェクトの増加としてサブシステムへ注入する (この増加したオブジェクトは、次回の交換において、のりしろ部分のオブジェクトとして削除される)。こうして、非分散システムの中のサブシミュレータは、自分が分散シミュレーションを行っていることを意識せずに、分散シミュレーションを構成することができる。

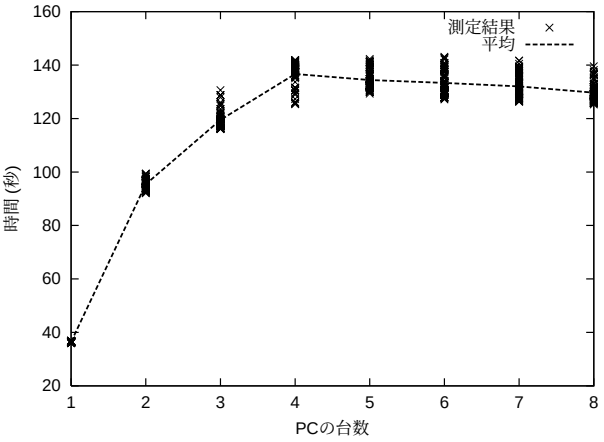

図 2: PC の台数に対する計算時間

4 実験

4.1 実験の内容

台数効果を測定するために、以下の実験を行った。

実験環境として、1 つの Pentium III 900MHz、256MB のメモリ、Linux 2.2.17 を搭載した PC を 8 台用意し、1 つのスイッチングハブを用いて、100BASE-TX のネットワークを構成した。

アプリケーションプログラムとして、簡単な災害シミュレーションを行うサブシミュレータを作成し接続した。サブシミュレータは、火災の延焼をシミュレーションするものと、市民の移動をシミュレーションするものと、シミュレーションの初期値を設定するサブシミュレータの 3 つを用意し、初期値として、阪神・淡路大震災が発生する前の神戸市長田区の地図 (1.5km×1.5km) を用意した。

1〜8 台の PC を用いて、$\frac{1.5\times(\text{PC の台数})}{8}$km × 1.5km の範囲のシミュレーションを行った。領域を短冊状に区切り、各 PC では 1.5/8km×1.5km の範囲に加え、隣接する区域のそれぞれ 10%をのりしろとしてシミュレーションを行った。1 台の PC で 1.5km×1.5km のシミュレーションを行うと、メモリのスワップにより正当な評価が難しくなるため、台数に応じてシミュレーション範囲を広げるようにした。

30 ステップのシミュレーションを、80 回ずつ行い、シミュレーションが終了するまでの時間を測定した。

4.2 結果と考察

図 2 の結果が得られた。また、これをもとに台数効果を計算したものが図 3 である。

図 2 では、台数に応じてシミュレーションの量をふやしているため、グラフが右肩上がりであれば台数効果が低く、水平であれば台数に比例したパフォー

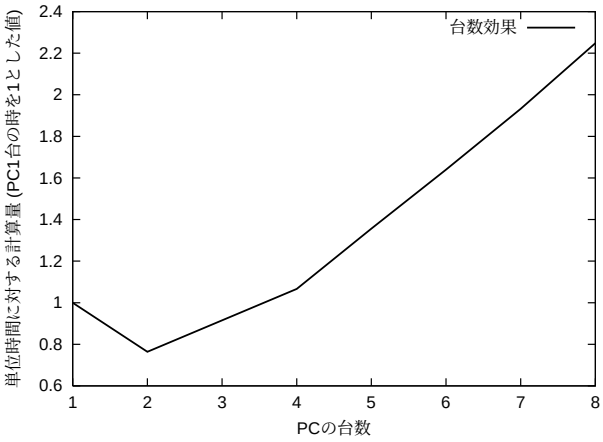

図 3: 台数効果

マンスを発揮していることになる。PC4 台まではグラフが右肩上がりになっており、台数効果は低い。しかし 4 台以降ではグラフはほぼ水平になっており、この範囲では、n 台の PC によるシミュレーションと m 台の PC によるシミュレーションを比べると、単位時間あたりの計算量はほぼ m/n 倍になる。シミュレーションの規模に対して計算機の台数を増やすことで、計算時間の増加を抑えることができていると言える。

しかし、PC1 台のみでシミュレーションを行った場合と比較すると、8 台に対して約 2.5 倍と、台数効果は低くなっている。これは、PC1 台のみでシミュレーションを行う場合、ネットワークを用いた通信や、のりしろ分のオーバーヘッドがないことが大きく影響していると考えられる。

5 関連研究

5.1 HLA

複数のシミュレータを結合してシミュレーションを行うシステムに、High Level Architecture[5, 6, 7] (HLA) がある。HLA は、本研究のシステムと同様、サブシミュレータ (federate と呼ばれる) をプラグインすることができる。しかし、HLA は離散イベントシミュレーション [8] に基づいており、スペースタイムをベースとした本研究とはモデルが異なる。

5.2 FUSS

同様に、複数のシミュレータを結合してシミュレーションを行うシステムとして、FUSS[9] が提案されているが、大規模なシミュレーション対象を想定しておらず、スケーラビリティに欠ける。

5.3 RoboCup-Rescue Ver.0 Kernel

筆者らが開発したプロトタイプシミュレータ (RoboCup-Rescue Ver.0 Kernel[10, 11]) も同様に、複数のシミュレータを結合してシミュレーションを行うシステムである。このシステムは、本研究における非分散システムに相当するシステムであり、シミュレーション対象に対してスケーラビリティがない。

6 将来への課題

動的拡張性 本システムでは、システムの初期化時にプラグインするサブシミュレータが決定され、シミュレーション開始後にサブシミュレータを抜き差しすることはできない。現実の災害において、対策本部における意思決定支援等に使用されることを想定すると、シミュレーションの最中に、それを見ている人間によってシミュレーションに新たな設定を導入できるインタラクティブ性が求められる。そのためには、サブシミュレータを動的にプラグインできるアーキテクチャが必要となる。

汎用性 のりしろ部分の交換等、いくつかのプログラムは、災害という特定のモデルに特化して作成されている。このため、災害以外のシミュレーション対象、あるいは、災害であっても異なるモデル化に基づく場合、それに応じたプログラムを作成する必要があり、作成すべきコードは少なくない。様々なシミュレーション対象に対して、それらのプログラムを簡単に作成できるフレームワークを作成する必要がある。

マルチエージェントシステムとしての洗練 シミュレーション対象の世界の中の、市民や消防士などのエージェントについて、積極的なサポートを行わなかった。このため、このシステムをマルチエージェントシステムとして利用するには、エージェントを扱うサブシミュレータをさらに開発する必要がある。

特に、災害シミュレーションにおいては、エージェント間の通信は、重要なシミュレーション要素であり、エージェント間通信のためのフレームワークを提供する必要がある。本研究のシステムは、地理的に離れた位置にあるオブジェクト間に直接の相互作用がない、あるいは相互作用の伝達に時間がかかることを前提にして成立している。エージェント間の通信は、地理的に離れているにもかかわらず、短時間で行われるため、特別な拡張、あるいはシミュレーションアーキテクチャ全体にかかわる見直しが必要になる。また、エージェント間の無線などを用いた通信は、メディア (空気、電線、電磁場等)、言語 (日本語、英語、方言等)、知識などによって成否がきまり、これらの概念を高度に抽象化した、通信の成否をシミュレーションするためのフレームワークが作成できる可能性がある。

また、エージェントが地理的に移動した場合、ある地域のシミュレーションを行っているプロセスと、その地域に存在するエージェントをコントロールするプロセスが、異なる計算機上に位置することになる。この場合に、モバイルエージェント等の技術によってプロセスを移動することによって、パフォーマンスが向上する可能性がある。

型付のモデル　現在のシステムではオブジェクトやプロパティに型がなく、ありえないオブジェクトやプロパティを自由に作成することができる。サブシミュレータを開発する際に、間違って、ありえないオブジェクトやプロパティを作ってしまうと、しばしばそのバグを発見することが困難である。このため、オブジェクトやプロパティに型をもたせ、静的あるいは動的にチェックできることが望ましい。

ユーティリティメソッド・クラスの追加　現在のシステムは必要最低限のプリミティブしか提供していない。このため、実際にサブシミュレータを開発するのは、大変な作業であり、開発を容易にするユーティリティの追加が必要である。具体的には、シミュレーション対象の世界のオブジェクトを、開発言語で扱えるオブジェクトとしてアクセスできる機能などが考えられる。

他言語のサポート　現在のシステムは Java のみをサポートしている。既に他の言語で開発されているシミュレータの接続を可能にするために、様々な言語をサポートすることが望ましい。

7　まとめ

複数のサブシミュレータをプラグインでき、大規模な対象に対して計算機を増やすことで、計算時間の増加を抑えることのできるシミュレーションアーキテクチャとカーネルを開発し、実験によって確認した。

参考文献

[1] 田所諭, 北野宏明他. ロボカップレスキュー. 共立出版, 2000.

[2] K. M. Chandy and R. Sherman. Space, time, and simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, Vol. 21, No. 2, pp. 53–57, March 1989.

[3] http://www.horb.org/.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[5] IEEE Std 1516-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.

[6] IEEE Std 1516.1-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification.

[7] IEEE Std 1516.2-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification.

[8] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of ACM*, Vol. 33, No. 10, pp. 30–53, October 1990.

[9] NODA, Itsuki. Framework of Distributed Simulation System for Multi-agent Environment. In *RoboCup 2000: Robot Soccer World Cup IV*. Springer, 2001.

[10] M. Ohta, T. Koto, I. Takeuchi, T. Takahashi and H. Kitano. Design and Implementation of the Kernel and Agents for the RoboCupRescue. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, 2000.

[11] 小藤哲彦, 竹内郁雄, 北野宏明. カーネルの通信と役割. 情報処理学会第 60 回全国大会, 2000.